Project#1

Languages accepted: C++, Python

In this project, you will get hands on experience about the (possibly) wisest search algorithm: **A* search**! In particular, you will write a program that solves the **eight-puzzle**. You will solve it using

1) **Uniform Cost Search**[1]
2) **A* with the Misplaced Tile heuristic.**
3) **A* with the Euclidean Distance heuristic.**

Notes: Complementary reading material on these algorithms is posted on Canvas (under Optional Reading).

Format: Your code should have an **object-oriented** design, as it **makes coding much easier** for you. In addition, it makes your code **simple and understandable** for yourself, for the TAs and for the instructor.

Before starting to code, please take some time to design. Whatever classes and methods you choose, make sure that your main "driver" code is general enough to look like the graph search algorithm pseudo code at the end of this page or the generic search algorithm pseudocode (tree search) on the slides and reading material.

To do this, you might want to implement a class called Problem, which might have instance variables like initial_state, goal_state, operators (a list of operators), etc. We later create instances of this class and give it to the search algorithm as the input parameter.

Optional: You might also want to implement a Tree class and a Node class. Each node object has a pointer to its parent node. We use the pointers to parent nodes in order to create the final solution (by following the parents of the goal node all the way up to the root of the tree). **Returning a final solution (i.e., the sequence of actions that actually led to the goal) is optional and has extra points.**

Important: Please choose **meaningful names** for your classes, methods, parameters, objects, and functions. This makes coding and understanding the code much easier. Please use comments as much as you can.

Important: The code should be kept **as general as possible**. In other words, your code should require only a modicum of effort to change to solve the 15-puzzle, or the 25-puzzle etc.

```
function GRAPH-SEARCH(problem) returns a solution, or failure
    initialize the frontier using the initial state of problem
    initialize the explored set to be empty
    loop do
        if the frontier is empty then return failure
        choose a leaf node and remove it from the frontier
        if the node contains a goal state then return the corresponding solution
        add the node to the explored set
        expand the chosen node, adding the resulting nodes to the frontier
            only if not in the frontier or explored set
```

[1] Note that Uniform Cost Search is just A* with $h(n)$ hardcoded to equal zero.

Of course, for the frontier you can use the pre-existing data structures like priority queue, etc.

Initial State: You can hardcode an initial state for testing purposes. But I want to be able to enter an arbitrary initial state. So something along the lines of the interface at the end of this document would be nice.

Using other sources: You may use some predefined utility routines, for example sorting routines or as said above, data structures like queue, list, etc. However, **I expect all the major code to be original.**
You must document any book, webpage, person other than your groupmates, or other resources you consult in doing this project (see the syllabus).

You may work in groups of at most 5 students. But **you may NOT share code with other groups**.

What you show to the TAs during demo sessions (demo sessions will be held the week after you turn in project1)**:**
You will need to demonstrate your code to the TA(s) and they can ask questions about any part of the code. As stated above, TA(s) should be able to enter arbitrary initial states and run the code with those states. It is a good idea to have a hard coded default initial state in your code too.

What you submit on Canvas
A zip file with:
1) Your code
2) A trace of the Euclidean distance A* on the following problem:

```
1 * 3
4 2 6
7 5 8
```

As noted above, finding/printing the correct sequence of actions/operators has extra points. *This is different from the trace here and needs keeping pointers to parent nodes*.
3) A two-three page report which summarize your findings. You might want to compare the algorithms on several random puzzles, summarize the results with graphs and make comments on the utility of the different heuristics etc. The only meaningful comparisons for the algorithms are time (number of nodes expanded) and space (the maximum size of the queue). See the Project1_ReportGuidelines.pdf for more information about the items you should include in your report (test cases, tables, graphs, etc.) .

You must list in your report all students in the group (groupmates) as well as the contribution of each student.
Note that ALL students in a group must submit the zip file (this zip file should be the same for all groupmates).
Also, ALL groupmates must be present in their demo session and each one of the students in the group will be asked questions and graded based on their answers.

You must keep the evolving versions of your code, so that, if necessary, you can demonstrate to the course staff how you went about solving this problem (in other words, we may ask you to prove that you did the work, rather than copy it from somewhere).

You can use a simple text line interface or a more sophisticated GUI (but don't waste time making it pretty unless you are sure it works, and you have lots of free time). However, your program should generate a trace like the one below, so that it can be tested.

Welcome to XXX (change this to your student ID) 8 puzzle solver.
Type "1" to use a default puzzle, or "2" to enter your own puzzle.
2
Enter your puzzle, use a zero to represent the blank
Enter the first row, use space or tabs between numbers **1 2 3**
Enter the second row, use space or tabs between numbers **4 8 0**
Enter the third row, use space or tabs between numbers **7 6 5**

Enter your choice of algorithm
Uniform Cost Search
A* with the Misplaced Tile heuristic.
A* with the Euclidean distance heuristic.

3

Expanding state
1 2 3
4 8 b
7 6 5

The best state to expand with g(n) = 1 and h(n) = 4 is…
1 2 3
4 8 5
7 6 b Expanding this node…

The best state to expand with g(n) = 2 and h(n) = 3 is…
1 2 3
4 8 5
7 b 6 Expanding this node…

.
. steps omitted here
.

The best state to expand with g(n) = 4 and h(n) = 1 is…
1 2 3
4 5 6
7 b 8 Expanding this node…

Goal!!!

To solve this problem the search algorithm expanded a total of XXX *(correct numbers should appear here)* nodes.
The maximum number of nodes in the queue at any one time: YYY.
The depth of the goal node was ZZZ.