Due date November 16, 2020
Late Submission 30% per day
Teams You can do the project individually or in teams of at most 3.
Teams must submit only 1 copy of the project via the team leader's account.
Purpose In this project, you will implement and analyse a heuristic search.

1 The χ-Puzzle

In this assignment you will implement and analyse a variety of search algorithms to solve the χ-Puzzle.

1.1 Rules of the Puzzle

The χ-Puzzle is a type of sliding-puzzle played on a wrapping board. The rules of the χ-Puzzle are the following:

1. The puzzle is an 2×4 board with 8 tiles (7 numbered and 1 empty).
2. *Regular moves:* Regular horizontal and vertical moves of sliding-puzzles are allowed and have a cost of 1.
3. *Wrapping moves:* If the empty tile is at a corner position, then the numbered tile at the other end of the same row can slide into it. These moves are more expensive than regular moves, and have a cost of 2.
4. *Diagonal moves:* If the empty tile is at a corner position, then the numbered tile diagonally adjacent to it inside the board, as well as the numbered tile in the opposed corner can be moved into it. These moves are more expensive than regular moves, and have a cost of 3.
5. The goal of the puzzle is to reach either one of the 2 goals below with the lowest cost.

1	2	3	4
5	6	7	0

1	3	5	7
2	4	6	0

1.2 Examples

Here are a few examples, to better understand the rules of the puzzle.

Example 1 Assume that the initial board is:

4	2	3	1
5	6	7	0

Then there are 5 possible moves:

1-2: regular moves - the 1 can move down into the 0 (with a cost of 1 – see rule #2 above) and the 7 can move right into the 0 (with a cost of 1 – see rule #2 above);

3: wrapping move - the 5 can move into the 0 (with a cost of 2 – rule #3 above).

4-5: diagonal moves - the 3 can move diagonally into the 0 (with a cost of 3 – see rule #4 above) and the 4 can can move diagonally into the 0 (with a cost of 3 – see rule #4 above);

.

These would lead to the following successor nodes:

4	2	3	0
5	6	7	1

4	2	3	1
5	6	0	7

4	2	3	1
0	6	7	5

4	2	0	1
5	6	7	3

0	2	3	1
5	6	7	4

Example 2 Assume that the initial board is

1	2	3	4
5	6	0	7

, then there are 3 possible moves: (3, 6, 7).

Example 3 Assume that the initial board is

1	0	3	4
5	6	2	7

, then there are 3 possible moves: (6, 1, 3).

2 Your Task

Implement a solution for the χ-board using the following 3 search algorithms:

1. Uniform Cost (UCS)
2. Greedy Best First (GBFS)
3. Algorithm $A^\star$ ($A^\star$)

For GBFS and $A^\star$, you must develop and experiment with 2 different heuristics h_1 and h_2. h_1 and h_2 should both be used with GBFS and with $A^\star$.

For the demo (see Section 4.1) you will run your code with the following naive heuristic, h_0.

$$h_0(n) = \begin{cases} 0, & \text{if } n = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ x_5 & x_6 & x_7 & 0 \end{bmatrix}, \text{regardless of the values of } x_1, x_2, x_3, x_3, x_4, x_5, x_6, x_7 \\ 1, & \text{otherwise} \end{cases}$$

The 2 heuristics that you developed (h_1 and h_2) cannot be equal to h_0.

2.1 Programming Environment

To program the project, you must use Python 3.8. In addition, you must use GitHub (make sure your project is private while developing).

2.2 The Input

Your code should be able to receive the path of an input file that contains initial puzzles. The input file will contain a sequence of lines, containing the values of the initial puzzle where each tile will be represented by a unique integer (0 to 7), where the zero will denote the empty tile. For example, the input file could contain:

```
1 0 3 7 5 2 6 4
0 3 7 5 2 6 4 1
1 0 3 7 5 2 6 4
```

puzzle #0
puzzle #1
puzzle #2

where puzzle #0 represents

1	0	3	7
5	2	6	4

There is no need to check the validity of the input file; you can assume that it will be correct.

2.3 Output

For each input puzzle and each search algorithm, your program should generate 2 output files: one for the solution path, and one for the search path. This means that for each line of the input file (see Section 2.2), your program should generate 10 text files:

Solution files	Search files
`#_ucs_solution.txt`	`#_ucs_search.txt`
`#_gbfs-h1_solution.txt`	`#_gbfs-h1_search.txt`
`#_gbfs-h2_solution.txt`	`#_gbfs-h2_search.txt`
`#_astar-h1_solution.txt`	`#_astar-h1_search.txt`
`#_astar-h2_solution.txt`	`#_astar-h2_search.txt`

where `#` is the puzzle number starting at zero.

For example, for puzzle #0 in of the input file in Section 2.2, you should generate:

Solution files	Search files
`0_ucs_solution.txt`	`0_ucs_search.txt`
`0_gbfs-h1_solution.txt`	`0_gbfs-h1_search.txt`
`0_gbfs-h2_solution.txt`	`0_gbfs-h2_search.txt`
`0_astar-h1_solution.txt`	`0_astar-h1_search.txt`
`0_astar-h2_solution.txt`	`0_astar-h2_search.txt`

2.3.1 Solution Files

Each solution file will contain the final solution found by the algorithm, or the string `no solution`. If a solution is found, the format of the file should be:

1. on line 1: 0␣0␣(two zero's) followed by the initial configuration (each tile separated by a space)
2. for each subsequent line: the token to move, a space, the cost of the move, a space, the new configuration of the board (each tile separated by a space)
3. the final line should contain the cost of the entire solution path, a space, then the execution time in seconds.

For example:

`0␣0␣4␣2␣3␣1␣5␣6␣7␣0`	*0, 0, then the initial puzzle*
`1␣1␣4␣2␣3␣0␣5␣6␣7␣1`	*move tile 1 with a cost of 1, which will lead to the new configuration*
...	...
`56␣1.2`	*cost of the solution path and execution time*

If your program cannot find a solution after 60 seconds, then it should output: `no solution` in both the solution and the search files.

2.3.2 Search Files

Each search file will contain the search path (the list of nodes that have been searched) by the algorithm. The search files should contain 1 line for each node searched. Each line should contain the $f(n)$, $g(n)$ and $h(n)$ values of the node separated by a space, followed by the configuration of the puzzle. For example:

`15␣10␣5␣4␣2␣3␣1␣5␣6␣7␣0`	$f(n) = 15, g(n) = 10, h(n) = 5$, state = 4 2 3 1 5 6 7 0
`9␣3␣6␣4␣2␣3␣0␣5␣6␣7␣1`	$f(n) = 9, g(n) = 3, h(n) = 6$, state = 4 2 3 0 5 6 7 1

If the value of $f(n)$, $g(n)$ or $h(n)$ are irrelevant for a specific search algorithm, just display its value as zero (`0`).

2.4 Analysis

Once your code is running, you will need to analyse and compare the performance of the algorithms and the heuristics. Generate a file with 50 random puzzles, then use this file as input to your 5 algorithms and compile the following data for each algorithm:

1. average & total length of the solution and search paths,
2. average & total number of `no solution`,
3. average & total cost and execution time,
4. optimality of the solution path.

Prepare a few slides explaining your heuristics and presenting and analysing the above data. You will present these slides at the demo (see Section 4.1.1). Your analysis should compare the data above across search algorithms and across heuristics.

2.5 Scaling Up

Once your analysis is done (see Section 2.4), take your best-performing algorithm and run it on a scaled-up version of the puzzle. This means that instead of using a 2×4 puzzle, increase the dimensions of the puzzle gradually and perform tests with random puzzles to see how the size of the puzzle influences the performance of your search. You can remove the 60-second time-out for this experiment. Note that on puzzles with more than 2 rows, the *wrapping moves* (applicable when the empty tile is at a corner position) allows the numbered tile at the other end of the same row as the empty tile to slide into it – but also allows the numbered tile at the other end of the same column as the empty tile to slide into it.

Prepare 1 or 2 slides presenting your analysis of the scaled-up puzzle, to present at the demo (see Section 4.1.1).

3 Competition (Just for fun)

Just for the fun of it, we will organize a competition to compare the solutions found by different teams with the same set of input puzzles. This competition will not count for points, we are just doing this for fun (and for the thrill of having your team publicly acknowledged on the Moodle page as the winner of the competition!).

For the competition, we will give all teams a file with random puzzles, and teams will be ranked based on the total cost of the solutions and execution time. More details on the competition will be posted on Moodle.

4 Deliverables

The submission of the project will consist of 2 deliverables: the code and a demo.

4.1 The Demos

You will have to demo your code to the TAs. Regardless of the demo time, you will demo the program that was uploaded as the official submission on or before the due date. The schedule of the demos will be posted on Moodle. The demos will consist in 2 parts: a small presentation of your analysis and a Q/A.

4.1.1 Presentation of your analysis

Prepare a few slides for a 5 minute presentation explaining your heuristics, your analysis and your scaled-up experiment (see Sections 2.4 and 2.5).

4.1.2 Q/A

No special preparation is necessary for the Q/A part of the demo. Your TA will give you a small input file of puzzles (the 2 version) and you will run all 5 algorithms with h_0. Your will generate the corresponding output files and upload them on EAS during the demo.

In addition, your TA will ask each student questions on the code/assignment, and the student will be required to answer the TA satisfactorily. Hence every member of team is expected to attend the demo. Your individual grade will be a function of your individual Q/A (see Section 5).

5 Evaluation Scheme

Students in teams can be assigned different grades based on their individual contribution to the project. Individual grades will be based on:

1. a peer-evaluation done after the deadline.
2. the contribution of each student as indicated on GitHub.
3. the Q/A of each student during the demo.

The team grade will be based on:

Code	functionality, correctness and format of the output files, design, programming style, ...	60%
Heuristics – h_1 and h_2	quality, originality, usability for $A^\star$ ...	10%
Demo – QA	clear answers to questions, knowledge of the program, ...	10%
Demo – Presentation	clarity and conciseness, depth of the analysis, presentation skills, time-management ...	20%
Total		100%

6 Submission

If you work in a team, identify one member as the team leader. The only additional responsibility of the team leader is to upload all required files from her/his account and book the demo on the Moodle scheduler. If you work individually, by definition, you are the team leader of your one-person team.

Submission Checklist

on GitHub:

- ☐ In your GitHub project, include a `README.md` file that contains on its first line the URL of your GitHub repository, as well as specific and complete instructions on how to run your program.
- ☐ November 19, make your GitHub repository public.

on EAS:

- ☐ Create a PDF of your slides, and name your slides `472_Slides2_ID1_ID2_ID3.pdf` where `ID1` is the ID of the team leader.
- ☐ Create one zip file containing all your code, your file of 50 random puzzles, the corresponding output files and the `README.md` file. Name this zip file `472_Code2_ID1_ID2_ID3.zip` where `ID1` is the ID of the team leader.
- ☐ Zip `472_Slides2_ID1_ID2_ID3.pdf` and `472_Code2_ID1_ID2_ID3.pdf` and name the resulting zip file: `472_Assignment2_ID1_ID2_ID3.zip` where `ID1` is the ID of the team leader.
- ☐ Have the team leader upload the zip file at: https://fis.encs.concordia.ca/eas/ as `Assignment 2`.

Have fun!