CS 240: Lab 1
8 Puzzle Problem

TAs: Soham Dahane, Dion Reji

Instructions

- This lab will be **graded**.
- Please read the problem statement and submission guidelines carefully.
- For any doubts or questions, please contact either the TA assigned to your lab group or one of the two TAs involved in making the lab.
- The deadline for this lab is **Thursday, 16 January, 5 PM** but solutions till 5:30 PM will be accepted. No submissions will be accepted after 5:30 PM.
- The submissions will be checked for plagiarism, and any form of cheating will be penalized.

8 Puzzle Problem

The 8 Puzzle is a classic sliding tile puzzle consisting of a 3×3 grid with tiles numbered from 1 to 8 and one blank space. The goal is to rearrange the tiles to match the target configuration by sliding one tile at a time into the blank space.

Example. Given an initial configuration:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & & 5 \\ 6 & 7 & 8 \end{bmatrix}$$

Transform it into the goal configuration:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & \end{bmatrix}$$

The common search algorithms used for solving this puzzle include:

1. **Breadth-First Search (BFS):** BFS explores all possible configurations level by level starting from the initial configuration, ensuring the shortest path is found in an unweighted state space.
2. **Depth-First Search (DFS):** DFS explores as deeply as possible along each branch before backtracking, leading to a solution.
3. **Dijkstra's Algorithm:** Dijkstra's algorithm finds the shortest path in a weighted graph by iteratively selecting the node with the smallest cost from the initial configuration to the current state.
4. **A* search:** You have seen A* search as an efficient graph traversal method to obtain the shortest path from a Start to Goal using some heuristic function. To this end, we use the following heuristic functions as discussed in class:

- **Displaced Tiles Heuristic:** The number of misplaced tiles from their destined position is used as the heuristic.
- **Manhattan Distance:** Read about Manhattan distance from here, if you haven't heard about it earlier! Now, use the sum of Manhattan distances of tiles from their destined positions as the heuristic to guide the search towards the solution efficiently.

In this lab, we will explore and compare the performance of different search and heuristic methods for solving the 8 Puzzle problem.

Problem Statement

You are given a python file `8puzzle.py`, which contains a `main` function and some basic functions with their signature. We will begin with some basic notation used throughout this document as well as in the reference python file given. Denote by `initial` and `goal` the initial and goal configurations of the 3×3 grid as a numpy array, respectively. The empty cell will be denoted by `0`. The blank has four different moves - *up, down, left, right.* We use the characters `'U','D','L','R'` to denote each of these moves respectively.

At each move, the empty cell can only move within the bounds of the 3×3 grid. Thus, at a given configuration, not all four moves are possible. Moreover, each move slides a single tile into the empty space. Also, the cost associated with each move of the empty cell is 1.

For this, you are required to implement BFS, DFS, Dijkistra and A* search algorithms and compare their performances.

Tasks to be Completed

In `8puzzle.py`, you are required to complete the following tasks:

Task 1. [10 marks] Complete the function `bfs(initial, goal)` by implementing Bread-First Search (BFS) to find the goal configuration from the initial configuration. This function should return a tuple consisting of:

- A list of characters from the set {`'U','D','L','R'`}, indicating the respective moves for transforming the initial configuration into the goal configuration.
- The number of nodes of the search graph expanded before termination.

Task 2. [10 marks] Complete the function `dfs(initial, goal)` by implementing Depth-First Search (DFS) to find the goal configuration from the initial configuration. This function should return a tuple consisting of:

- A list of characters from the set {`'U','D','L','R'`}, indicating the respective moves for transforming the initial configuration into the goal configuration.
- The number of nodes of the search graph expanded before termination.

Task 3. [16 marks] Complete the function `dijkistra(initial, goal)` by implementing Dijkistra's algorithm to find the goal configuration from the initial configuration. This function should return a tuple consisting of:

- A list of characters from the set {`'U','D','L','R'`}, indicating the respective moves for transforming the initial configuration into the goal configuration.
- The number of nodes of the search graph expanded before termination.
- The total cost for transforming the initial configuration into the goal configuration following the sequence of moves found by your algorithm.

Task 4. [17 marks] Complete the function `astar_dt(initial, goal)` by implementing A* Search to find the goal configuration from the initial configuration, using the Displaced Tiles Heuristic. This function should return a tuple consisting of:

- A list of characters from the set {`'U','D','L','R'`}, indicating the respective moves for transforming the initial configuration into the goal configuration.
- The number of nodes of the search graph expanded before termination.
- The total cost for transforming the initial configuration into the goal configuration following the sequence of moves found by your algorithm.

Task 5. [17 marks] Complete the function `astar_md(initial, goal)` by implementing A* Search to find the goal configuration from the initial configuration, using the Manhattan Distance Heuristic. This function should return a tuple consisting of:

- A list of characters from the set {`'U','D','L','R'`}, indicating the respective moves for transforming the initial configuration into the goal configuration.
- The number of nodes of the search graph expanded before termination.
- The total cost for transforming the initial configuration into the goal configuration following the sequence of moves found by your algorithm.

Additional Notes

- Ensure that your implementation can solve any solvable 8 Puzzle configuration provided as input.
- Make sure you remove all `print` statements as it will interfere with auto-grading. A penalty will be imposed for not adhering to this instruction.
- You are **not allowed** to change the name or signature of the functions already given to you in `8puzzle.py`. This will affect auto evaluation. If you change them, the respective functions won't be evaluated. However, you are free to write new helper functions and classes.
- Do not import any other package unless allowed by the TAs in charge of the lab.
- Follow the instructions strictly to avoid evaluation issues.

Submission

- Submissions should be made on Moodle. Submit the python file renamed as `rollnumber1_rollnumber2.py` (the "b" in roll number should be in small case).
- Penalty will be imposed on wrong file naming.
- The hard deadline for submission is 5:30 pm. No submission after that will be evaluated.
- Only one person per team should submit their solution.

Evaluation

This lab will be evaluated in two phases- Auto-grading and Viva. For auto-grading, each of the tasks to be completed will be evaluated independently and the respective marks are as shown in the **Tasks to be Completed** section.

30 marks will be given during viva to be conducted next week.