Middle East Technical University Department of Computer Engineering

CENG 462

Artificial Intelligence

Fall '2024-2025

Homework 1

Due date: 4 November 2024, Monday, 23:55

1 Objectives

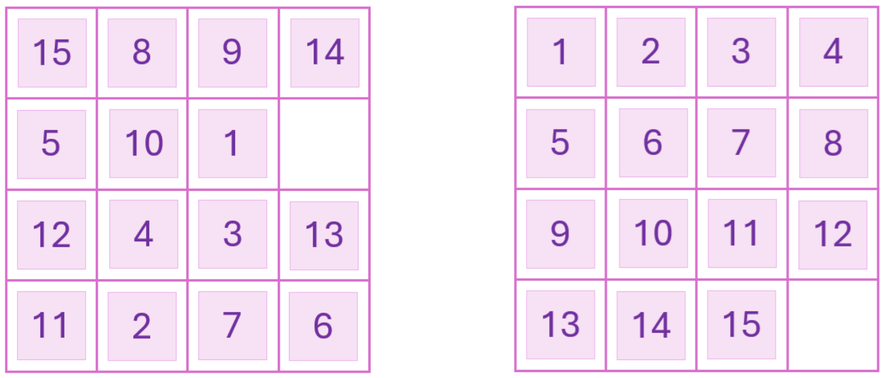
This assignment aims to improve your understanding of search algorithms, specifically Depth-First Iterative Deepening and A* methods while providing practical experience with uninformed and informed search techniques.

2 Problem Definition

In this assignment, you are going to solve a given N-puzzle for N=15 using the A* and Depth-First Iterative Deepening algorithms.

The 15-puzzle consists of a 4x4 board with 15 sliding tiles and one blank space. The objective is to reach a specific goal configuration from a given initial configuration by sliding tiles into the blank space.

However, in this version of the 15-puzzle, in addition to moving tiles that are adjacent to the blank space, you can also move tiles that are **two spaces away** from the blank space.
Here is an example of initial and goal configurations for 15-puzzle:



(a) initial state (b) goal state

Figure 1: Example configurations as initial and goal states for 15-puzzle

In this example, starting from the initial state, the available actions include moving up(13), down(14), and right(1). Additionally, unlike the standard 15-puzzle, you can also move two spaces up(6) and two spaces to the right(10).

3 Specifications

- You are going to implement A* and Depth-First Iterative Deepening algorithms in Python.
- You will use the **total Manhattan distance** as the heuristic function, which estimates the total distance each tile is from its goal position. In addition to the heuristic, the total cost function $f(n)$ is calculated as:

  $$f(n) = g(n) + h(n)$$

  Where:

  - $g(n)$ is the actual cost to reach the current state from the initial state. This includes the cost of moving tiles.
    - Moving to an adjacent tile (up, down, left, or right) has a cost of **2 units**.
    - Jumping to a tile two spaces away (either horizontally or vertically) incurs a higher cost of **3 units**.
  - The **total Manhattan distance** will be used as the heuristic function. The Manhattan distance between two tiles is defined as the sum of the absolute differences between their row and column positions. For a tile located at position (r_i, c_i) on the current board, and its goal position being (r_g, c_g), the Manhattan distance is calculated as:

    $$\text{Manhattan distance} = |r_i - r_g| + |c_i - c_g|$$

    The total Manhattan distance $h(n)$ for a board configuration is the sum of the Manhattan distances for all tiles, excluding the blank tile.
- The task will be given from standard input and the result will be printed to standard output.
- Input will consist of:
  - the method you are going to use for the given task ("A*" or "DFIDS" for Depth-First Iterative Deepening Search)
  - M as the maximum total estimated cost allowed in the method for A*, and the maximum depth for DIFDS, (we add a maximum depth limit in DFIDS to prevent the search from going too deep)
  - the initial configuration of the board,
  - the goal configuration of the board.
- Output will consist of "SUCCESS" or "FAILURE" stating whether the goal is reached or not respectively. If the solution exists, all configurations on the path from the initial configuration to the goal configuration will be printed as well.
- While expanding a node you are required to try to move **up**, **down**, **left**, **right**, **two-up**, **two-down**, **two-left**, **two-right** in that order (Moving up means sliding the tile below the blank upward, similar for other directions.)

- When searching for the state with the minimum cost you will select the **first** one for tie-breaking.

IMPORTANT NOTE: Last two items are crucial for the black-box evaluation. Please avoid any violation of them, in order not to lose any points redundantly.

4 Sample I/O

Input 1:

```
A*
50
1 6 2 3
5 _ 8 4
9 10 7 11
13 14 15 12
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 _
```

Output 1:

```
SUCCESS

1 6 2 3
5 _ 8 4
9 10 7 11
13 14 15 12

1 _ 2 3
5 6 8 4
9 10 7 11
13 14 15 12

1 2 _ 3
5 6 8 4
9 10 7 11
13 14 15 12

1 2 3 _
5 6 8 4
9 10 7 11
13 14 15 12

1 2 3 4
5 6 8 _
9 10 7 11
13 14 15 12

1 2 3 4
5 6 _ 8
9 10 7 11
13 14 15 12

1 2 3 4
5 6 7 8
9 10 _ 11
13 14 15 12

1 2 3 4
5 6 7 8
9 10 11 _
13 14 15 12

1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 _
```

Input 2:

```
DIFDS
50
1 _ 2 4
5 6 3 7
9 10 15 8
13 14 12 11
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 _
```

Output 2:

```
SUCCESS

1 _ 2 4
5 6 3 7
9 10 15 8
13 14 12 11

1 2 _ 4
5 6 3 7
9 10 15 8
13 14 12 11

1 2 3 4
5 6 _ 7
9 10 15 8
13 14 12 11

1 2 3 4
5 6 7 _
9 10 15 8
13 14 12 11

1 2 3 4
5 6 7 8
9 10 15 _
13 14 12 11

1 2 3 4
5 6 7 8
9 10 15 11
13 14 12 _

1 2 3 4
5 6 7 8
9 10 15 11
13 14 _ 12

1 2 3 4
5 6 7 8
9 10 _ 11
13 14 15 12
```

```
1 2 3 4
5 6 7 8
9 10 11 _
13 14 15 12

1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 _
```

Input 3:

```
DIFDS
15
2 1 3 4
5 6 7 8
9 10 11 12
13 14 15 _
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 _
```

Output 3:

```
SUCCESS

2 1 3 4
5 6 7 8
9 10 11 12
13 14 15 _

2 1 3 4
5 6 7 8
9 10 11 _
13 14 15 12

2 1 3 4
5 6 7 8
9 10 _ 11
13 14 15 12

2 1 _ 4
5 6 7 8
9 10 3 11
13 14 15 12

2 _ 1 4
5 6 7 8
9 10 3 11
13 14 15 12

_ 2 1 4
5 6 7 8
9 10 3 11
13 14 15 12
```

```
1 2 _ 4
5 6 7 8
9 10 3 11
13 14 15 12

1 2 3 4
5 6 7 8
9 10 _ 11
13 14 15 12

1 2 3 4
5 6 7 8
9 10 11 _
13 14 15 12

1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 _
```

Input 4:

```
DIFDS
10
15 14 13 12
11 10 9 8
7 6 5 4
3 2 1 _
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 _
```

Output 4:

```
FAILURE
```

5 Regulations

1. **Programming Language:** You must code your program in Python3.
2. **Implementation:** You have to code your program by only using the functions in standard module of Python. Namely, you **cannot** import any module in your program. The only exception for this rule is the `copy` module which may be useful with the `deepcopy` function to cope with aliasing list problem in Python.
3. **Late Submission:** No late submission is allowed.
4. **Cheating: We have zero tolerance policy for cheating**. People involved in cheating (any kind of code sharing and codes taken from internet included) will be punished according to the university regulations.
5. **Evaluation:** Your program will be evaluated automatically using "black-box" technique so make sure to obey the specifications. A reasonable timeout will be applied according to the complexity of test cases in order to avoid infinite loops due to an erroneous code.

6. **Submission:** Submission will be done via OdtuClass system. You should upload a **single** Python file named **<your-student-id>_hw1.py** (i.e. e1234567_hw1.py).