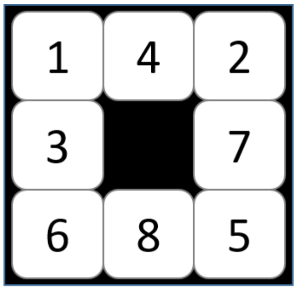
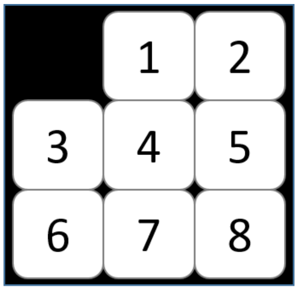
Target Topic: Iterative Deepening Search (IDS), Greedy Best-first Search

Case Study : 8-PUZZLE





INPUT: A permutation of {0, .. , 8} arranged in 3x3 format. 0 represents the hole.

OUTPUT: The sequence of moves required to reach the goal state, with the minimum number of moves

Provided materials:

- 8puzzle_ids_template.py : the incomplete program for IDS.
  - 8puzzle_ids_template_EZer.py is the easier version, for students struggling with programming
- 8puzzle_gbfs_template.py : the incomplete program for Greedy Best-first Search
  - Simple_Priority_Queue.py : a simple priority queue class required for this search technique
- 8-puzzle testcases.zip from Week 1

Task:

1. Study to make sure that you understand what problem of BFS that the IDS addresses.

2. Study the provided 8puzzle_ids_template.py program for what are required to complete the program.
   - As IDS utilizes Depth-first Search (DFS), so a function for DFS is required for implementing IDS. However, the EZer version includes a complete DFS function.

3. Complete the program so that it solves the 8-puzzle problem with IDS technique.
   - The first step is probably to find the minimum number of moves to solve.
   - Once the above step is accomplished, then attempt to make the program compute the optimal sequence of moves (with the Print-path method that was already studied in Week 1)

4. Test the program with the provided test cases. Note the limitation of the program.

5. Study the provided simplePriorityQueue.py for how to utilize the Simple_Priority_Queue class. A few examples are included at the bottom of the program file.

6. In the 8puzzle_gbfs_template.py file, develop a heuristic function for a given 8-puzzle pattern. Then add its corresponding value as a field of class “state”. Every newly created successor state should have this field updated before inserting into successor list.

7. Create a priority queue in the gbfs function. This priority queue facilitates dequeuing state that has smallest heuristic value, rather than the “first-come” state that is dequeued by Breadth-first Search.

8. Complete the program so that it solves the 8-puzzle problem with Greedy Best-first Search technique.

9. Test the program with the provided test cases. A correct one will solve the case very fast and for the provided test case, optimally correct if solved. However, make sure that you understand why the Greedy Best-first Search does not guarantee an optimal solution.