Experiment 1: Search

Please read the rules for assignments on the course web page. Of course, **do not share code**. Acknowledge any help you received and any sources you used. Please use the online website （Zhihuishu） to turn in the assignment.

1 Introduction

In this assignment, you will implement A* algorithm in Python, and apply it to the two problems below. The helper code you need is provided on the course web page, you just need to fill in the missing parts.

The layout of the helper code:

• node.py - The implementation of the Node class. (Do not modify this file!)

• problems.py - The partial implementation of the FifteensNode and SuperqueensNode classes.

• search.py - The skeleton of the A* algorithm.

• test.py - A helper script that runs some tests.

You need to code up the function Astar in search.py, and the following methods of the FifteensNode and SuperqueensNode classes:

• is_goal

• generate_children

• evaluate_heuristic

You can use built-in functions & data structures, and the standard library, but you **cannot** use/import anyone else's code or any package that is not in the Python standard library. You can use test.py to test your code:

$ python -m unittest test.py

However, your code will be tested on some secret instances of the problems; therefore, you should be careful about the boundary cases.

Note that you should not necessarily expect your algorithms to solve every instance (difficult instances may require too much time or memory; that does not mean that you did not solve the experiment problem correctly). Of course, your code is expected to output the right answer when it outputs something.

You should submit problems.py and search.py to the Zhihuishu assignment "Experiment 1: Search", and you should upload them directly without zipping.

2 Fifteens Puzzle

The first problem that you will solve using A* is the classic fifteens puzzle (the four-by-four version of the eights puzzle studied in class) where you can move any horizontally or vertically adjacent tile into the empty slot.

For example, here is the goal state:

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

The following states are one move away from the goal state:

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

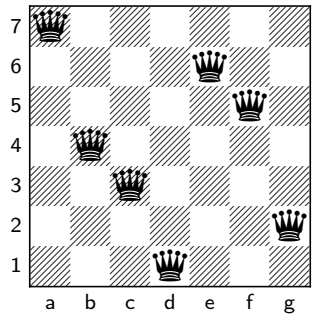
1	2	3	4
5	6	7	8
9	10	11	
13	14	15	12

Every move has a cost of 1, and of course, since you are using A*, your program should find the optimal (lowest-cost) solution. In principle, there may be more than one optimal solution; your program is just expected to give one of these optimal solutions—it does not matter which one. You can use the heuristics discussed in the lectures, or, if you want, you can try to design something even better (the book has some more detail).

3 Superqueens Puzzle

Consider a modified chess piece that can move like a queen, but also like a knight. We will call such a piece a “superqueen” (it is also known as an “amazon”). This leads to a new “superqueens” puzzle. We formulate the puzzle as a constraint optimization problem: each row and each column can have at most one superqueen (that’s a hard constraint), and we try to minimize the number of pairs of superqueens that attack each other (either diagonally or with a knight’s move).

For example, the following is an optimal solution for a 7 by 7 board: there are 3 total attacks (two diagonal, one a knight’s move).



The cost of a node should be the number of pairs of superqueens that attack each other so far; you can set the heuristic to 0. You are not required to implement sophisticated variable ordering (that is, you can always consider the variables in the same order), constraint propagation, etc. (although you are welcome to do so).

4 Examples

4.1 Constructing Instances

```
In [1]: from problems import FifteensNode
        input_str = """
         1  2  3  4
         5  6  7  8
         9 10  0 11
        13 14 15 12
        """
        fifteens_root = FifteensNode(input_str=input_str)
        print(fifteens_root)
```

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

```
In [2]: from problems import SuperqueensNode
        superqueens_root = SuperqueensNode(n=7)
        print(superqueens_root)
```

```
. . . . . . .
. . . . . . .
. . . . . . .
. . . . . . .
. . . . . . .
. . . . . . .
. . . . . . .
```

4.2 Goal States

```
    ...
```

```
In [4]: print(fifteens_node)
        print(fifteens_node.is_goal())
```

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

True
```

```
    ...
```

```
In [6]: print(superqueens_node)
        print(superqueens_node.is_goal())
```

```
 Q  .  .  .  .  .  .
 .  .  .  Q  .  .  .
 .  .  .  .  Q  .  .
 .  .  .  .  .  .  Q
 .  Q  .  .  .  .  .
 .  .  Q  .  .  .  .
 .  .  .  .  .  Q  .
True
```

4.3 Node Expansions

```
In [7]: print('Parent Node:')
        print(fifteens_root)
        for i,child in enumerate(fifteens_root.generate_children()):
            print('\nChild',i+1,':')
            print(child)
```

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


Child 1 :
  1  2  3  4
  5  6     8
  9 10  7 11
```

```
 13 14 15 12


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


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


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

4.4 Heuristic Functions

```
In [8]: print(fifteens_node)
        print(fifteens_node.evaluate_heuristic())
```

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

0
```

4.5 A* Algorithm

```
In [9]: from search import Astar
        fifteens_path = Astar(fifteens_root)
        for i,node in enumerate(fifteens_path):
            print('Node',i+1,':')
            print(node)
```

```
Node 1 :
  1  2  3  4
```

```
  5  6  7  8
  9 10    11
 13 14 15 12

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

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

```
In [10]: superqueens_path = Astar(superqueens_root)
         print('Node 1 :')
         print(superqueens_path[0])
         print('\nNode',len(superqueens_path),':')
         print(superqueens_path[-1])
```

```
Node 1 :
 .  .  .  .  .  .  .
 .  .  .  .  .  .  .
 .  .  .  .  .  .  .
 .  .  .  .  .  .  .
 .  .  .  .  .  .  .
 .  .  .  .  .  .  .
 .  .  .  .  .  .  .

Node 8 :
 .  .  .  Q  .  .  .
 Q  .  .  .  .  .  .
 .  .  .  .  .  .  Q
 .  .  .  .  .  Q  .
 .  Q  .  .  .  .  .
 .  .  Q  .  .  .  .
 .  .  .  .  Q  .  .
```