COMP3702 Artificial Intelligence
Semester 2, 2022
Tutorial 2

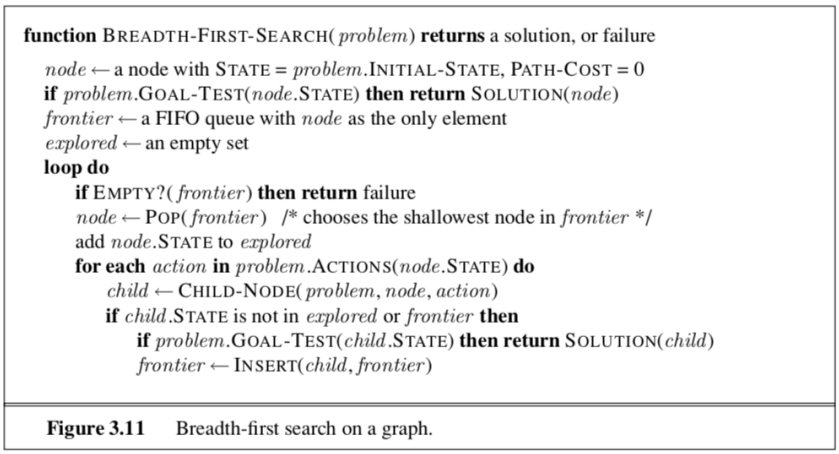
Before you begin, please note:

- Tutorial exercises are provided to help you understand the materials discussed in class, and to improve your skills in solving AI problems.
- Tutorial exercises will not be graded. However, you are highly encouraged to do them for your own learning. Moreover, we hope you get the satisfaction from solving these problems.
- The skills you acquire in completing the tutorial exercises will help you complete the assignments.
- You'll get the best learning outcome when you try to solve these exercises on your own first (before your tutorial session), and use your tutorial session to ask about the difficulties you face when trying to solve this set of exercises.

Exercises

Exercise 2.1. Please implement the Breadth First Search (BFS) and Depth First Search (DFS) algorithms to solve the 8-puzzle problem.

The pseudocode for BFS is shown in Figure 1



```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier)  /* chooses the shallowest node in frontier */
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child ← CHILD-NODE(problem, node, action)
      if child.STATE is not in explored or frontier then
        if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
        frontier ← INSERT(child, frontier)
```

Figure 3.11 Breadth-first search on a graph.

Figure 1: Pseudocode for Breadth-First-Search on a graph (Source: Russell & Norvig textbook)

Alternatively, see P&M's discussion of BFS: `https://artint.info/2e/html/ArtInt2e.Ch3.S5.SS1.html`.

Exercise 2.2. Please test your implementation of BFS and DFS with the following initial and goal states:

- From 1348627_5 to 1238_4765
- From 281_43765 to 1238_4765
- From 281463_75 to 1238_4765

In each test for each algorithm, please output:

(a) The sequence of actions to move from the initial to the goal state.

(b) The number of steps to solve the problem.

(c) The time your implementation takes to solve the problem.

Based on the results of (b) and (c), please explain the relation between the time and the number of steps that each algorithm takes to solve the problem. Is the relation you found in-line with the complexity results of the algorithms? If they are not in-line, please find out and explain why.

Exercise 2.3. Suppose there is no solution to the 8-puzzle problem, i.e., there is no way the 8-puzzle can move from the given initial to goal configurations.

(a) Will BFS be able to output the right answer (i.e., no solution)? How about DFS?

(b) We can actually identify if a given 8-puzzle problem is solvable or not without search, which is via the concept of parity (see the appended material on inversion and parity). Please implement parity computation, so that your program can identify if a problem is solvable without performing search.

Exercise 2.4. Imagine the tiles have been tampered with and it is no longer equally easy to move the tiles in each direction. Suppose that the cost for 'moving the empty tile' is as follows:
cost = {'up':1, 'down':2, 'left':3, 'right':4}

(a) Will Uniform Cost search find the least cost path for this problem?

(b) Compare Uniform Cost Search and BFS when the weight of all edges in the state graph are equal.

Inversion and Parity

We can identify if a given 8-puzzle problem is solvable or not without search by analysing the number of *inversions* between the input state and the goal state. A pair of tiles form an inversion if the values on the tiles are in reverse order of their appearance in the goal state. If the number of inversions is odd it is not solvable; while if it is even it is solvable. This is represented by the *parity*.

To calculate the parity of an 8-puzzle game instance, we first fix the ordering, e.g. left-right, up-down order. We then define the following variables:

- tile-i is the tile with number i on it
- $N(s, i) = \#$inversions of tile-i in state s
  - i.e. the number of out of order pairs of tiles with respect to tile-i
  - This is equal to $\#$tiles with smaller number than tile-i that appears after tile-i
- N(s) is $\#$inversions in state s: $\sum_{i=1}^{n} N(s, i)$
- Parity(s) $= N(s) \bmod 2$
  - Indicates whether the number of inversions of state s is odd or even. If it's even, the parity will be 0 and indicates that the 8-puzzle is solvable

Claim: The parity is *invariant* (does not change) under any legal move of the blank tile.
Proof:

- Any horizontal move will not change the parity.
- Any vertical move changes N by an even number.

Consequence: An 8-puzzle is only solvable if the parity of both the start and goal states are the same.

Example: Odd parity

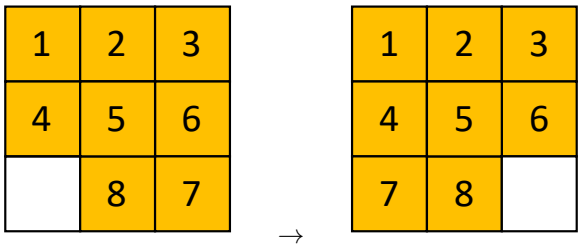


Figure 2: Given state and goal state

For the example in Figure 2, we can write the goal state as 1,2,3,4,5,6,7,8 (ignoring the blank tile), and the initial state as 1,2,3,4,5,6,8,7. In this case, we have 1 inversion (7 & 8 are swapped), and therefore the parity is odd and the puzzle is not solvable.

Example: Even parity

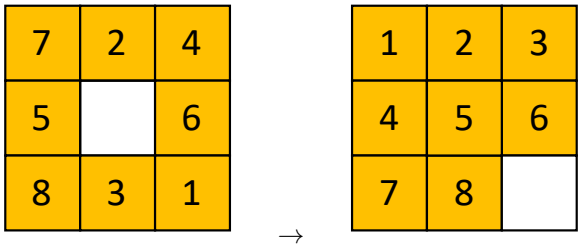


Figure 3: Given state and goal state

For the example in Figure 3, we have a total of N(s) = 16 inversions:
N(s,7) = 6;
N(s,2) = 1;
N(s,4) = 2;
N(s,5) = 2;
N(s,6) = 2;
N(s,8) = 2;
N(s,3) = 1;
N(s,1) = 0;

See the following figure for calculating each of these.

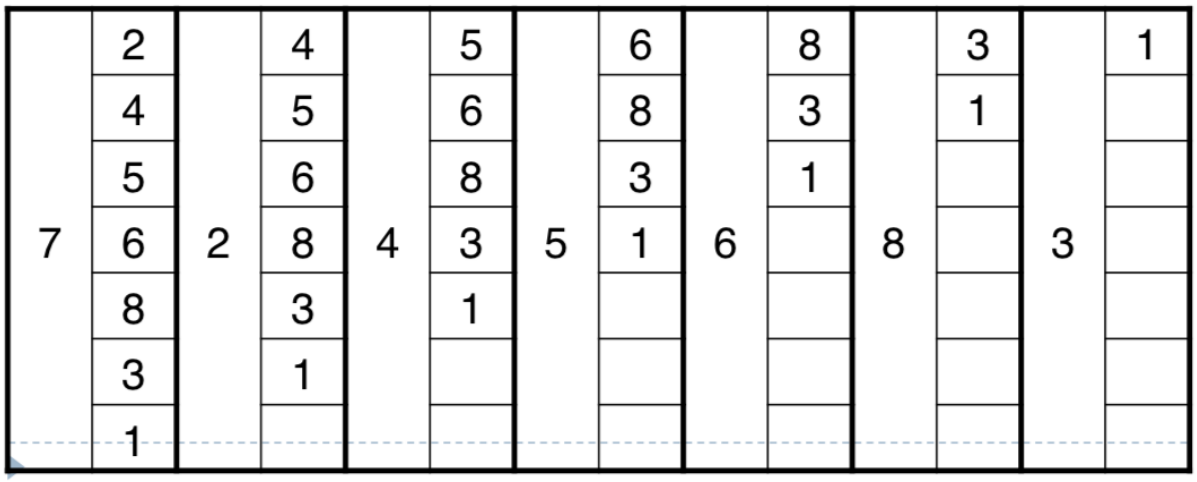


Figure 4: Calculating the number of inversions of each tile for the example in Figure 3

Therefore the parity is even and the puzzle is solvable!