AI Homework 2

Steven DiCarlo snd7nb
Yaser Qazi yq4du

$\boxed{1.4}$ State has two elements: the priority queue (pq) and the list of seen_puzzles.

The priority queue contains sequences of puzzles, where each puzzle follows from the previous one. The priority is determined by how many steps this sequence is plus the number of tiles still out of place in the final puzzle of the sequence.

In each transition between states, the top priority sequence is removed from pq. Each possible puzzle that can be generated in one move from the last puzzle in the sequence is then generated (call them successors). For each successor, if it is the goal state, then that chain is the solution. Otherwise, for each successor not in seen_states, it is added to seen_states and the sequence ending in it is added to pq.

This is adequate because every possible puzzle that can be generated from the original will eventually be checked. Once pq is empty, every possible puzzle will be in seen_states, none of which were the solution, so there could not have been any solution

$\boxed{1.5}$ Proving correctness: If it returns something, was it right?

Case 1: returns solution

· If it returns a solution, that means it could find a chain of moves that led to the goal state, which means it was correct

Case 2: returns unsolvable:

· At the beginning, we check if the puzzle is solvable by counting the number of inversions. Inversions are when if you list the puzzle in order (left to right top to bottom, like reading a book), every time a bigger number precedes a smaller one is an inversion. We claim that if you start with a puzzle with even inversions (even puzzle), every move will result in another even puzzle.

Case A: row move:

If the move shifts a tile along its row, then the order of the tiles is the same, so there is the same number of inversions, so it is still even.

$$\begin{array}{ccc} 1 & 2 & 3 \\ 4 & & 5 \\ 8 & 6 & 7 \end{array} \rightarrow \begin{array}{ccc} 1 & 2 & 3 \\ & 4 & 5 \\ 8 & 6 & 7 \end{array}$$

Case B: column move:

If the move shifts a tile along its column, then that tile moves relative to two tiles in the order (in this example 2 goes from before 3 and 4 to after them). This means that the number of inversions can only change by -2, 0, or 2 with each column move, so if it started even, it will stay even.

$$\begin{array}{ccc} 1 & 2 & 3 \\ 4 & & 5 \\ 8 & 6 & 7 \end{array} \rightarrow \begin{array}{ccc} 1 & & 3 \\ 4 & 2 & 5 \\ 8 & 6 & 7 \end{array}$$

Since the goal state has an odd number of inversions (7), that means it can't be obtained from any even puzzle, so if our algorithm returns unsolvable it is correct.

$\boxed{1.6}$ Proving completeness: If there is a solution, will we find it?

In order for there to be a solution, it must be generated from a sequence of moves from the original puzzle. Our algorithm generates all possible puzzles from the original, and eventually checks all of them to see if it is a goal state. Therefore if there is a solution, our algorithm will eventually be able to find it.

3. By the termination of BFS, dist(s,x) >= d[x], where d[x] signifies the shortest path from the root node to x, dist(s,x) and dist(s,x) is the distance found between nodes s and x, where s is our root node. This is because dist(s,x) is the length of some path from s to x, while by definition, d[x] is the shortest path from s to x, so the equality must hold true. Using this statement, we can prove the fact that by the termination of BFS, d[x] = dist(s,x) will hold true for any and all x in our search space. We will prove this statement by contradiction.

We assume that there is some vertex v where d[v] ≠ dist(s,v). Let v be the closest node from s such that d[v] ≠ dist(s,v). Since we know that dist(s,v) >= d[v] must hold true, according to the definition laid out above, dist(s,v) > d[v].

Let's consider the shortest path from s to v, and let's have a node u such that the edge (u,v) is the last edge on the shortest path from s to v. Since the length of our shortest path to v is d[v], and since (u,v) is the last edge, d[v] = d[u] + 1. Since u is closer to s than v, and by our choice of v, since v was the closest node such that d[v] ≠ dist(s,v), d[u] = dist(s,u). We can therefore conclude the following equalities:

1. dist(s,v) > d[v]
2. d[v] = d[u] + 1
3. d[u] = dist(s,u)
4. dist(s,v) > d[u] + 1 (from 1 and 2)
5. dist(s,v) > dist(s,u) + 1 (from 3 and 4)

We will now consider three cases to find a contradiction to statement number 5. We consider the status of v when u is first explored by our BFS. The three cases:

1. v has not yet been discovered
    a. Since we have defined an edge (u,v), v will be discovered during the exploration of u. This means that dist(s,v) = dist(s,u) + 1, a direct contradiction to statement number 5
2. v has already been discovered and explored
    a. This case means that v was enqueued and dequeued before u was enqueued. This means that, by definition, dist(s,v) < dist(s,u), which contradicts statement number 5
3. v has been discovered but not yet explored
    a. Let there be an arbitrary node w that discovered v. This discovery must have happened before the exploration of u, otherwise it would have discovered v. This means w must have been explored before u was explored, which means that since w was enqueued before u, dist(s,w) < dist(s,u), so dist(s,w) < dist(s,u) + 1. Since w discovered v. dist(s,v) = dist(s,w) + 1. We can therefore write dist(s,v) < dist(s,u) + 1, which contradicts statement 5

QED