LAB 6 Informed Search

In **informed search** strategy we use problem-specific knowledge beyond the definition of the problem itself so that we can find solutions more efficiently than can an uninformed strategy.

1.1 Greedy Best First Search

In Best-first search algorithm a node is selected for expansion based on an **evaluation function**, $f(n)$. The evaluation function is construed as a cost estimate, so the node with the *lowest* evaluation is expanded first. **Greedy best-first search** tries to expand the node that is closest to the goal, on the grounds that this is likely to lead to a solution quickly. Thus, it evaluates nodes by using just the heuristic function; that is, $f(n) = h(n)$.

```
Function best ()
{
     queue = [];     // initialize an empty queue
     state = root_node;   // initialize the start state
     while (true)
     {
          if is_goal (state)
               then return SUCCESS
          else
          {
               add_to_front_of_queue (successors (state));
               sort (queue);
          }
          if queue == []
               then report FAILURE;
          state = queue [0]; // state = first item in queue
          remove_first_item_from (queue);
     }
}
```

1.2 A* Search

The most widely known form of best-first search is called **A* search**. It evaluates nodes by combining $g(n)$, the cost to reach the node, and $h(n)$, the cost to get from the node to the goal:

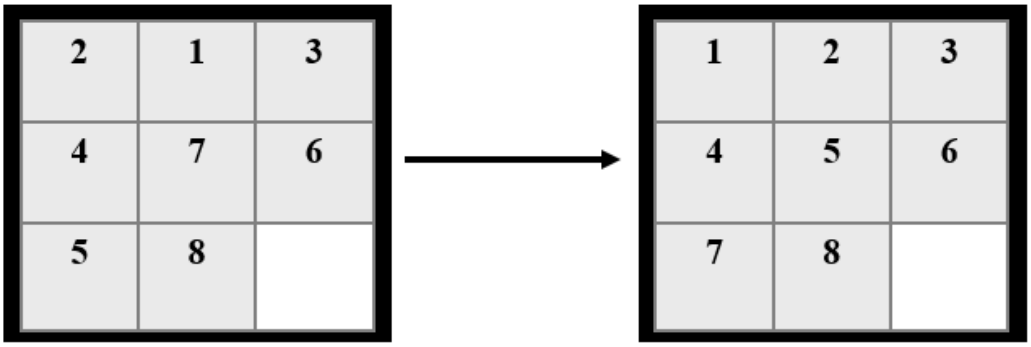
$$f(n) = g(n) + h(n)$$

Since $g(n)$ gives the path cost from the start node to node n, and $h(n)$ is the estimated cost of the cheapest path from n to the goal, we have $f(n)$ = estimated cost of the cheapest solution through n. Thus, if we are trying to find the cheapest solution, a reasonable thing to try first is the node with the lowest value of g(n) + h(n).

1.3 Lab Tasks

Exercise 6.1.

Using state representation, goal_test() and operators from previous labs, implement Greedy Best First Search to solve *8-Puzzle* problem. You should write a function *GreedySearch(state)* that accepts any initial state and returns the result.



The first heuristic we consider is to count how many tiles are in the wrong place. We will call this heuristic, $h_1(n)$. An improved heuristic, $h_2(n)$, takes into account how far each tile had to move to get to its correct state. This is achieved by summing the **Manhattan distances** of each tile from its correct position. (Manhattan distance is the sum of the horizontal and vertical moves that need to be made to get from one position to another). Implement both the heuristic to solve the given 8-puzzle problem.

Exercise 6.2.

Solve above problem using A* Search algorithm.