FUNDAMENTALS OF ARTIFICIAL INTELLIGENCE - CS161
Programming Assignment 4 - Due Tuesday, February 5, 11:59 p.m.

DO NOT CONSULT ANY OUTSIDE REFERENCES OR SOURCES FOR THIS ASSIGNMENT

Write a LISP program that solves the Eight Puzzle using Iterative-Deepening-A* (IDA*). It consists of a three by three frame containing eight numbered tiles and one empty or "blank" space. Any tile horizontally or vertically adjacent to the blank can slide into the blank position. The task is to find a shortest sequence of moves that map the initial state to the goal state shown below. If a problem instance is not solvable, your program should return the atom "fail". The maximum depth of any eight-puzzle instance is 31 moves. Your top-level function, called IDA*, should take just one argument representing a state and return either "fail", or a shortest list of tiles moved to solve the problem.

```
0 1 2
3 4 5
6 7 8
```

Use the Manhattan distance heuristic function. It is computed by summing, for each tile except the blank, the distance of that tile in grid units from its goal position. The easiest and most efficient way of computing manhattan distance is to use a table that lists the manhattan distance between each pair of positions. Assume that every move costs one unit.

The simplest representation for a state is a nine-element list where each position in the list corresponds to a particular position in the frame, and the element in a given position represents the number of the tile currently occupying that position. Let the labels of the positions be as shown in the goal state, where 0 represents the blank. Thus, the goal state above would be represented by the list (0 1 2 3 4 5 6 7 8).

The easiest way to represent the operators internally is by compiling a list that gives, for each possible position of the blank, the list of positions that the blank could move to from there. The following is such a list: ((1 3) (0 2 4) (1 5) (0 4 6) (1 3 5 7) (2 4 8) (3 7) (4 6 8) (5 7)). The most convenient way to represent moves to the user is by giving the number of the tile moved in each case. Thus, your program's input should be an initial state represented as a list of tiles, and the value returned should be a shortest list of tiles to be moved to solve the problem instance, or the atom "fail" if the problem is unsolvable.

Concentrate on writing a number of small functions rather than large functions. Do not be overly concerned with efficiency at this point. For example, a certain amount of overhead is incurred by using a list representation instead of arrays. Your function comments must precisely state what value is returned as a function of the arguments and distinguish tile numbers from tile positions. Write your functions in a bottom up order and thoroughly test each function before using it in a higher function. Be sure to rigorously test your program. Note that only half of all initial states are solvable.

You should get started early on this. In order to handle the complexity, I suggest that you first implement and debug depth-first iterative-deepening, then eliminate generating the parent of a node as one of its children, and then modify your program into IDA* by adding the heuristic evaluation. Note that each of these simpler programs should work correctly, but will take longer to run. Thus, test the simpler versions on simpler problem instances. Note that only half of all problem instances are solvable. In particular if you swap the positions of two physical tiles, then you can no longer reach the orginal state.