CS 486/686—Introduction to Artificial Intelligence

Assignment 2

Spring 2001
Department of Computer Science
University of Waterloo

Due: *Tuesday, June 12 at 2:00pm* (before class)
Worth: 10% of final grade (3 questions worth 1%, 4%, and 5%, plus a 2% bonus question.)
Instructor: Dale Schuurmans, DC1310, x3005, dale@cs.uwaterloo.ca

---

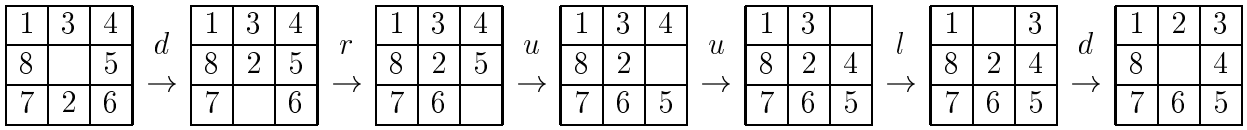
Note: Submit three Scheme functions in files called: randep.scm, ids.scm, and idas.scm (and, if you choose, the answer to the bonus question in a fourth file called idaplus.scm).

The eight-puzzle

The eight puzzle consists of a 3×3 board with numbered squares $1, ..., 8$ and a special *blank* square. The blank can be swapped with a square that is immediately up, down, left or right of it. The goal state is given by the configuration:

1	2	3
8		4
7	6	5

An instance of the eight-puzzle problem is specified by giving an initial state. The objective is to find a shortest sequence of moves that takes the initial state to the goal state. For example:



In this assignment you will implement two procedures for solving eight-puzzle problems: an iterative deepening search, `ids`, and an iterative deepening A^* search, `idas` (plus a random initial state generator, `randep`).

Representation

To implement these search procedures you will need to represent states and actions in Scheme. For this assignment you will represent states by a list where the ith location of the list contains the number of the square in the ith location of the board. The board locations will be indexed

ℓ_1	ℓ_2	ℓ_3
ℓ_4	ℓ_5	ℓ_6
ℓ_7	ℓ_8	ℓ_9

Thus, the goal state,

1	2	3
8		4
7	6	5

, will be represented by a list `(1 2 3 8 9 4 7 6 5)`, where the blank square is represented by a `9`.

- The action *move-up* ("`u`") is implemented by swapping `9` with the number three positions to the left of it in the list (if possible).
- The action *move-down* ("`d`") is implemented by swapping `9` with the number three positions to the right (if possible).
- The action *move-right* ("`r`") is implemented by swapping `9` with the number one position to the right of it in the list, as long as `9` is not in the ℓ_3, ℓ_6 or ℓ_9 position.
- The action *move-left* ("`l`") is implemented by swapping `9` with the number one position to the left, as long as `9` is not in the ℓ_1, ℓ_4 or ℓ_7 position.

Part 1 (Generating eight-puzzle problems—1%)

Write a Scheme function "`randep`" that takes one argument, d, and generates a random initial state by starting with the goal state and taking d random (legal) moves from the goal. Your procedure should return a list that represents a state.

Part 2 (Iterative deepening search—4%)

Write a Scheme function "`ids`" that takes two arguments s and g, where s is an initial state and g is a goal state, and returns a shortest list of moves `(`m_1 m_2 ... m_k`)` that takes the initial state s to the goal g. Each move m_i must be one of `l`, `r`, `u`, or `d`. Specifically you must implement an *iterative deepening search* to do this.

Thus,
`(ids '(1 3 4 8 9 5 7 2 6) '(1 2 3 8 9 4 7 6 5))` should return `(d r u u l d)`.

Part 3 (Iterative deepening A^* search—5%)

Write a Scheme function "`idas`" that takes the same two arguments as above and returns a shortest list of moves `(`m_1 m_2 ... m_k`)` that takes the initial state s to the goal g. However, for this part you must implement this using an *iterative deepening A^* search*. *IDA** requires a heuristic function $\hat{h}(s)$ for estimating the number of steps remaining from a state s to a goal g. Use the following heuristic:

$$\hat{h}(s) = \sum_{\text{pieces}} \text{distance from piece's location in } s \text{ to target location in } g$$

where the distance is calculated by determining the shortest (u,d,l,r) path from a piece's location in s to its goal location in g.

Bonus! (2%)

Come up with an effective way to prune additional states in `idas`. Write a Scheme function `idaplus` that incorporates your ideas and runs significantly faster than `idas`.