CS 486/686—Introduction to Artificial Intelligence

Assignment 2

Fall 2001
Department of Computer Science
University of Waterloo

Due: *Monday, Oct 15 at midnight* (night before Tuesday lecture)
Worth: 10% of final grade
(3 questions worth 1%, 4%, and 5%, plus a 2% bonus question.)
Instructors: Dale Schuurmans, DC1310, x3005, dale@cs.uwaterloo.ca
Vlado Keselj, DC2133, x3378, vkeselj@cs.uwaterloo.ca

Note: Submit three Java files: `Randep.java` (containing public class `Randep`, which contains the function `randep`), `Ids.java` (containing public class `Ids`, which contains the function `ids`), and `Idas.java` (containing public class `Idas`, which contains the function `idas`). If you choose to answer the bonus question, submit the fourth file `Rubik.java` (containing public class `Rubik`, which contains the functions `randep` and `ids`).

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:

$$\begin{array}{|c|c|c|}\hline 1&3&4\\\hline 8&&5\\\hline 7&2&6\\\hline\end{array} \xrightarrow{d} \begin{array}{|c|c|c|}\hline 1&3&4\\\hline 8&2&5\\\hline 7&&6\\\hline\end{array} \xrightarrow{r} \begin{array}{|c|c|c|}\hline 1&3&4\\\hline 8&2&5\\\hline 7&6&\\\hline\end{array} \xrightarrow{u} \begin{array}{|c|c|c|}\hline 1&3&4\\\hline 8&2&\\\hline 7&6&5\\\hline\end{array} \xrightarrow{u} \begin{array}{|c|c|c|}\hline 1&3&\\\hline 8&2&4\\\hline 7&6&5\\\hline\end{array} \xrightarrow{l} \begin{array}{|c|c|c|}\hline 1&&3\\\hline 8&2&4\\\hline 7&6&5\\\hline\end{array} \xrightarrow{d} \begin{array}{|c|c|c|}\hline 1&2&3\\\hline 8&&4\\\hline 7&6&5\\\hline\end{array}$$

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 Java. For this assignment you will represent states by an array of integers where the ith location of the array contains the number of the square in the ith location of the board. The board locations will be indexed

ℓ_0	ℓ_1	ℓ_2
ℓ_3	ℓ_4	ℓ_5
ℓ_6	ℓ_7	ℓ_8

Thus, the goal state,

1	2	3
8		4
7	6	5

, will be represented by an array {`1, 2, 3, 8, 0, 4, 7, 6, 5` }, where the blank square is represented by a `0`.

- The action *move-up* ("`u`") is implemented by swapping `0` with the number three positions to the left of it in the list (if possible).
- The action *move-down* ("`d`") is implemented by swapping `0` with the number three positions to the right (if possible).
- The action *move-right* ("`r`") is implemented by swapping `0` with the number one position to the right of it in the list, as long as `0` is not in the ℓ_2, ℓ_5 or ℓ_8 position.
- The action *move-left* ("`l`") is implemented by swapping `0` with the number one position to the left, as long as `0` is not in the ℓ_0, ℓ_3 or ℓ_6 position.

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

Write a Java function "`randep`" with the following signature:

```
public static int[] randep(int d)
```

which, for an argument `d` (`d`≥ 0), generates a random initial state, by starting with the goal state, and taking d random (legal) moves from the goal. Your function should return an array that represents the state.

Part 2 (Iterative deepening search—4%)

Write a Java function "`ids`" with the following signature:

```
public static char[] ids(int[] s, int[] g)
```

The function takes two arguments `s` — an initial state, and `g` – 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 characters: `l`, `r`, `u`, or `d`. Specifically you must implement an *iterative deepening search* to do this. You can assume that a solution exists.

Thus,

```
int[] s = {1,3,4,8,0,5,7,2,6};
int[] g = {1,2,3,8,0,4,7,6,5};
char[] r = Ids.ids(s, g);
```

should return

```
r = { 'd', 'r', 'u', 'u', 'l', 'd' };
```

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

Write a Java function "`idas`" with the following signature:

```
public static char[] idas(int[] s, int[] g)
```

The precondition and postcondition is the same as for `ids`. 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%)

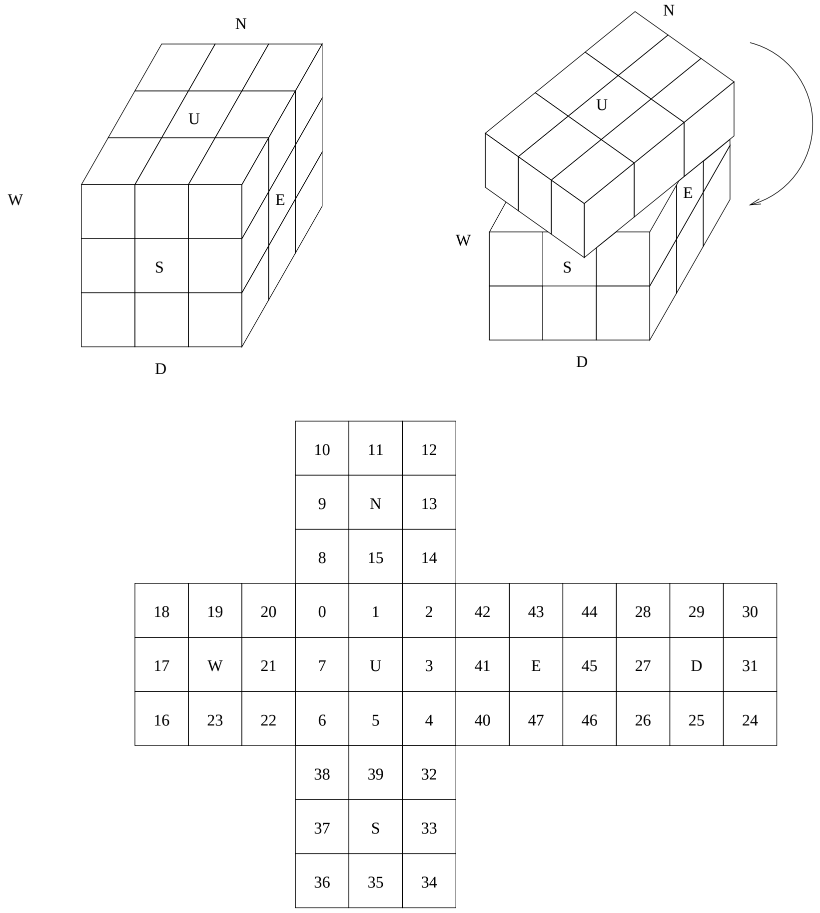
Write the `randep` and `ids` functions for Rubik's cube. The functions must be defined in the file `Rubik.java`, i.e., in the class `Rubik`, and they must have signatures:

```
public static int[] randep(int d)
```

and

```
public static String[] ids(int[] s, int[] g)
```

A Rubik's cube is a cube divided into 27 smaller cubicles ($3 \times 3 \times 3$). Cubicles can be moved in the fashion shown in the following figure: (You can also take a look at the Java applet in the course web site: `http://www.student.math.uwaterloo.ca/~cs486/rubik/`)



The sides can be marked as U (for up), N (for north), W (for west), S (for south), E for (east), and D (for down). The center squares are fixed, and all the other squares are movable. The figure also illustrates how the movable squares are enumerated. In this way all positions are enumerated, as well as all visible sides of movable squares. A state is represented as an integer array of length 48. An action is a permutation of the array. There are 18 actions: U1 (rotate the upper side for 90° in clockwise direction), U2 (rotate the upper side for 180°), and U3 (rotate the upper side for 270°), and similarly N1, N2, N3, W1, W2, W3, D1, D2, D3, S1, S2, S3, E1, E2, and E3 (rotation is always done in clockwise direction). Each of the actions can be represented as a permutation of numbers $\{0, 1, \ldots, 47\}$. In the goal state, each square is at the same-numbered position, i.e., the goal permutation is: $0, 1, 2, \ldots, 47$.

Each string in the output of the function `ids` has to be one of the following: `U1`, `U2`, `U3`, `N1`, ..., `E3`, as listed above.

Try to reduce the branching factor in the function `ids`. If you do so, explain this in form of a Java comment at the beginning of the file `Rubik.java`.