CS 486/686—Introduction to Artificial Intelligence

Assignment 2

Spring 2003
School of Computer Science
University of Waterloo

Due: *Wednesday, June 11 at 23:59:59 local time*
Worth: 10% of final grade
(4 questions worth 1%, 3%, 4% and 2%, plus a 2% bonus question.)
Instructors: Relu Patrascu, DC2127, x3299, rpatrasc@cs.uwaterloo.ca
Dale Schuurmans, DC1310, x3005, dale@cs.uwaterloo.ca

Note: Submit four Java files:
`Randep.java` containing the public class `Randep`, which contains the function `randep`;
`Ids.java` containing the public class `Ids`, which contains the function `ids`;
`Idas.java` containing the public class `Idas`, which contains the function `idas`; and
`Idasfast.java` containing the public class `Idasfast`, which contains the function `idasfast`.
If you choose to answer the bonus question, submit a fifth file `Rubik.java` containing the 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 any square that is immediately above, below, to the left, or to the 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 a random initial state generator, `randep`, and three procedures for solving eight-puzzle problems: an iterative deepening search, `ids`; an iterative deepening A^* search, `idas`; and a faster version of IDA^* search, `idasfast`.

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 *i*th location of the array contains the number of the square in the *i*th location of the board. The board locations will be indexed

ℓ_0	ℓ_3	ℓ_6
ℓ_1	ℓ_4	ℓ_7
ℓ_2	ℓ_5	ℓ_8

Thus, the goal state,

1	2	3
8		4
7	6	5

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

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

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

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

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

which, for an argument `d`≥ 0, generates a random initial state by starting with the goal state and taking *d* random (legal) moves from the goal. The function should return an array representing the state it reaches.

Part 2 (Iterative deepening search—3%)

Write a Java function "`ids`" with the 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 $\{m_1,\ m_2,\ \ldots m_k\ \}$ 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 a solution exists.)

Thus,

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

should return

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

Part 3 (Iterative deepening A^* search—4%)

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

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

The precondition and postcondition for this procedure are the same as for `ids`. However, for this part you must implement the problem solver by using an *iterative deepening* A^* *search*. Recall that 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

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

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.

Part 4 (Faster IDA^* search—2%)

Write a Java function "`idasfast`" with the signature

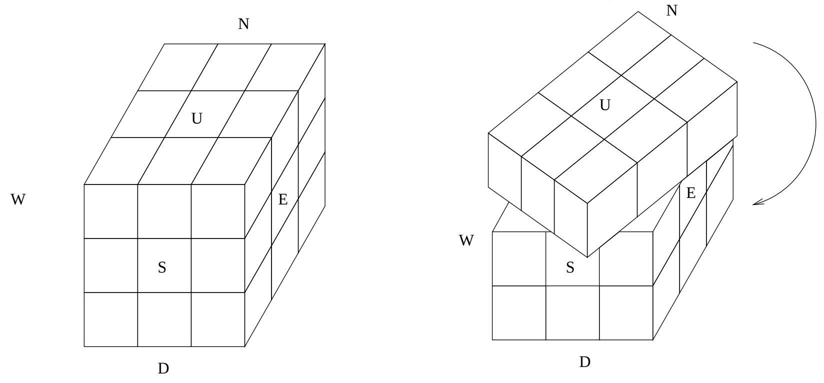
```
public static char[] idasfast(int[] s, int[] g)
```

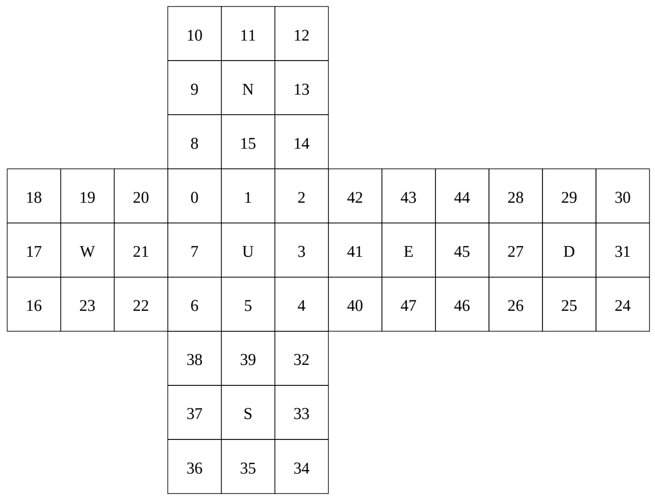
Implement an efficient method for pruning additional states that results in a significantly faster search than `idas`. Come up with an idea for doing this on your own, and clearly document the method you use.

Bonus! (2%)

Write `randep` and `ids` for the Rubik's cube problem. The functions must be defined in the file `Rubik.java`, in the class `Rubik`, with the same signatures as before.

Rubik's cube is a large cube that has been subdivided into 27 smaller cubies ($3 \times 3 \times 3$). The cubies can be moved in sub-groups by twisting the faces of the cube in the manner shown in the figure. (You can also take a look at the Java applet in the course website `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 cubies are fixed, and all other visible cubies are movable. The figure illustrates how the visible cubie faces are identified. A state is represented as an integer array of length 48, and an action is a permutation of this array. There are 18 actions: U1 (rotate the upper side 90° clockwise), U2 (rotate the upper side 180° clockwise) and U3 (rotate the upper side 270° clockwise); similarly N1, N2, N3, W1, W2, W3, D1, D2, D3, S1, S2, S3, E1, E2 and E3 (where rotation is always done clockwise). Each of the actions can be represented as a permutation of numbers $\{0, 1, \ldots, 47\}$. In the goal state, each visible cubie face is in its correct position. That is, the goal permutation is $0, 1, 2, \ldots, 47$.

The random problem generator `randep` should generate an initial state by starting at the goal and, given input `d`≥ 0, taking `d` random moves and returning the state that results.

Given an initial state, the function `ids` should return a sequence of moves (i.e., a string of symbols from `U1`,`U2`,...,`E3`) that transforms the intial state into the goal state.