ECE 479/579

Principles of Artificial Intelligence

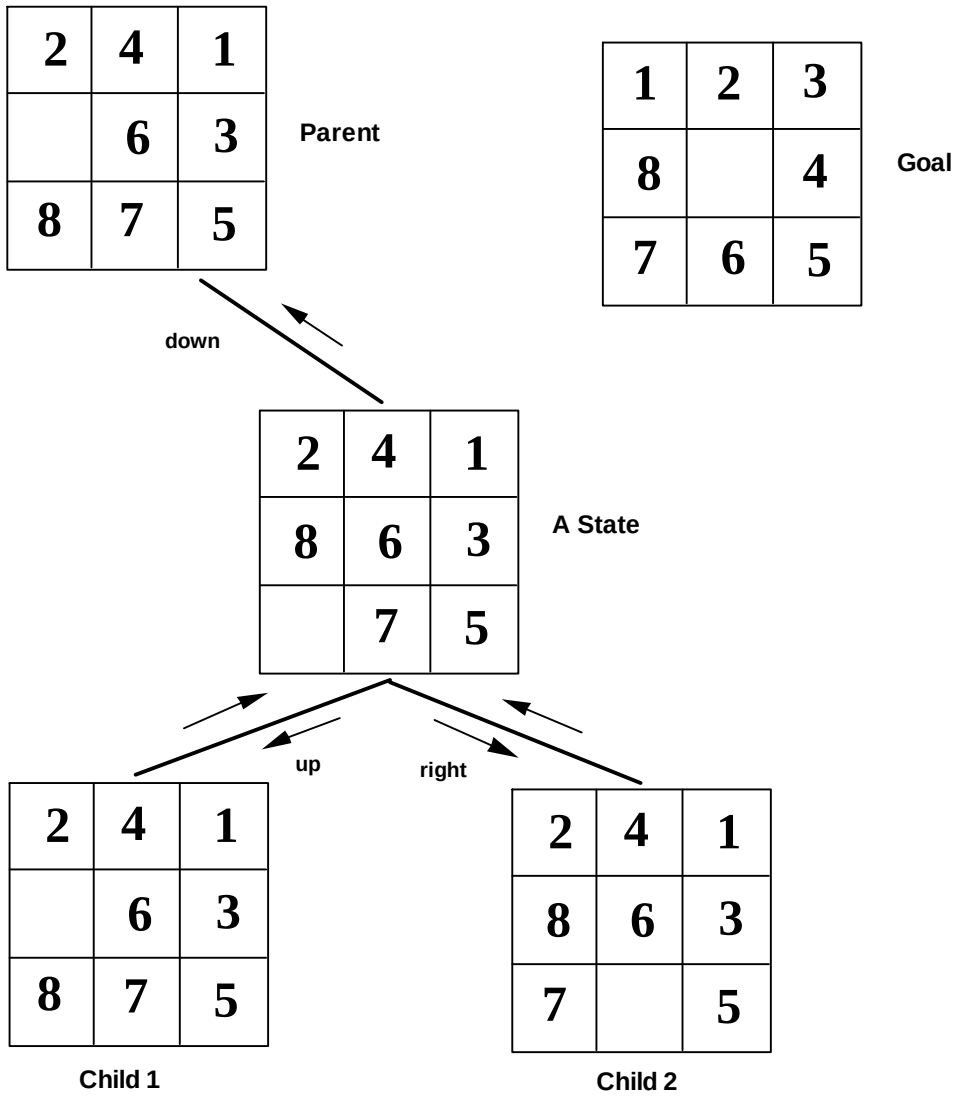
Dr. Marefat

PROJECT # 3

“Heuristic Search”

In this assignment, you are required to solve the **8-puzzle problem** that has been discussed in class using **Best-First** graph search algorithm implemented in **JAVA**. You have been provided with a set of interfaces and abstract classes representing the node and puzzle solver class behavior. You are required to make two classes "EightPuzzleNode.java" and "EightPuzzleSolver.java", which extend the `AbstractNode` and `AbstractPuzzleSolver` classes respectively.

PART I (40 points) – **`EightPuzzleNode.java`** `extends AbstractPuzzleNode`
The figure below shows one possible state in an eight puzzle. Suppose the board for each state is represented by a list of 9 elements which indicate the value for each cell in the puzzle (c1 c2 c3 c4 c5 c6 c7 c8 c9). For example, (2 4 1 8 6 3 0 7 5) corresponds to the board value for the state in the figure. Each child can be represented by a similar list of 9 elements, and there could be at most 4 children for each state (up, down, left, right). The parent (or path back) for a state can also be represented by the 9 element list that shows its board value. We also associate two values with each state, gval and hval. `hval` is set equal to the number of tiles out of place (as compared with the goal) or any other feasible heuristic. `gval` is equal to the gval of parent plus 1 (gval at start is zero).



Therefore, the total representation for a node consists of the following:

(i) A list/array of integers representing its corresponding board.
(ii) A list/array of 4 or less node objects.
(iii) A node corresponding to the current best parent.

The `EightPuzzleNode` class should have a concrete implementation for the following functions:

(a) Write a function **`expand`**, (current state of the game), will compute its possible children (nodes) and return them as an array of nodes.

(b) Two states are equal if their board representations are the same/equal. Write a function called **`equals`** that, given any other state, will return `true` if the two states are equal, and `false` if the two states are not equal.

You are allowed to have additional functionality built into `EightPuzzleNode` class, which will be helpful or beneficial for you in part II of this assignment. But the above `expand` and `equals` functions should be implemented in accordance with the specifications given above. You may look at the `Node` interface provided, for the above-mentioned functions.

PART II (60 points) – `EightPuzzleSolver.java` `extends AbstractPuzzleSolver`

Assume the initial status of the 8-puzzle is an input to your `EightPuzzleSolver` class. Fig 1 shows one possible starting configuration.

2		**4**
6	**3**	**7**
1	**5**	**8**

Fig 1. The initial status of the 8-puzzle problem

Successful termination condition should be as shown in Fig 2.

1	**2**	**3**
8		**4**
7	**6**	**5**

Fig 2. The termination condition

The `EightPuzzleSolver` class should have a concrete implementation for the following functions:

- **(a)** Write a function **`orderStates`** that when it is given an array of nodes, will return another array containing the same nodes as its elements, but ordered in the ascending order of the value of the sum, gval + hval.
- **(b)** Write a function **`solve`** that when called, will return back an array of nodes in proper order (from start state to final/goal state), which represent the solution to the 8-puzzle.

NOTE: The parent of a node (at any time, there is only one node as the value of parent) MUST be updated by using the `setParent` function in the `AbstractNode` class. The `setParent` function adds the previous parent to a list of past parents. This will be checked for grading, to make sure that back pointers have been updated correctly. You may take a look at the `PuzzleSolver` interface provided, for the above-mentioned functions.

Your best-first graph search should solve the problem using the heuristic F = g(n) + h(n), where g(n) is the cost of path to node n, and h(n) may be any of the following:

1. h() = No. of displaced tiles
2. h() = The manhattan distance
3. $h() = \begin{cases} \left\lfloor \frac{man}{2} \right\rfloor & if\ \ man\%2 = 0\ (man\ is\ even) \\ \left\lfloor \frac{man}{4} \right\rfloor & if\ \ man\%2 = 1\ (man\ is\ odd) \end{cases}$ *where* $\lfloor\ \rfloor$ *represents the flooring function*

and 'man' is the manhattan distance. As you can notice this function is still "admissible" but is not consistent (please refer to section 4.1 and 4.2 of the book). *"If the consistency condition on F is satisfied, then when A* expands a node n, it has already found an optimal path to n"* (Pg. 151, Artificial Intelligence, A new synthesis; Nils J. Nilsonn). This will not be true for this function and hence back pointers may need to be updated.

Specifications

Your program is required to adhere to the specifications listed here or it cannot be graded.

PROGRAM NAME: The program names for Part I and II of the assignment should be "EightPuzzleNode.java" and "EightPuzzleSolver.java" respectively. Please comment your code neatly and provide `javadoc` comments for all class variables and functions of your classes.

COMPILATION PROCEDURE: Your programs will be tested with multiple test cases on shell.ece.arizona.edu. Please make sure that your programs compile correctly, making use of the provided jar file (in this case the command will be "`javac –classpath .:./heuristic.jar` <YourFile.java>") or by making use of the source code itself (javac *.java). The latter case assumes that all the java files are in the same directory.

The package name for the interfaces and abstract classes provided is "`ece479.projects.heuristicSearch`", and hence the import statements (if any needed) in your program will be something like "`import ece479.projects.heuristicSearch.AbstractNode`".

INPUT: A test class has also been provided to test you programs. For example, the initial state example given above is represented as "`int[] initState = new int[]{2,0,4,6,3,7,1,5,8}`". Note that the number 0 is used to represent the position of the blank tile. Also note that the tiles are placed in the list in **row-major** order. Given this state the following lines of code in the test class should lead to a solution:

```
EightPuzzleSolver eps = new EightPuzzleSolver(initState, null);
Node[] disSolution = eps.solve("dis");
Node[] manSolution = eps.solve("man");
Node[] manSolution = eps.solve("man2");
```

The argument to the function `solve` is the heuristic to be used. Use the `String` "dis" for number of displaced tiles, `String` "man" for manhattan distance and `String` "man2" for the third heuristic mentioned before.

OUTPUT: As mentioned before the solve function will return an array of nodes in proper order, from the start to the end/goal node. These node objects should have correct values for their parent (possibly null), children (never null), g and h values. The test class provided can be used for testing whether solution techniques have been correctly implemented or not. An example node with a parent of (2 0 4 6 3 7 1 5 8) and two children: (2 0 4 6 3 7 1 5 8) and (2 4 7 6 3 0 1 5 8), with a current state of (2 4 0 6 3 7 1 5 8), a heuristic value of 8 and a path cost of 1, is shown as follows (this is the way the test class provided prints out the solutions):
((2 4 0 6 3 7 1 5 8) ((2 0 4 6 3 7 1 5 8) (2 4 7 6 3 0 1 5 8)) (2 0 4 6 3 7 1 5 8) 1 8).

TURNIN
All program files need to be turned in as follows on shell.ece.arizona.edu:
`turnin ece479 proj3 EightPuzzleNode.java EighPuzzleSolver.java`