RECITATION 6: Homework discussion
17 March 2003

At present, the code you are using searches the starting board, then its child boards, then its grandchildren, etc. Here is a search tree representing this kind of search from a particular starting board. Notice that the goal (solved) state for their puzzle is:

1	2	3
8		4
7	6	5

But the goal state we want to reach in the homework is this one. Don't worry about the difference, since the code we gave you is already set up for the right goal state.

1	2	3
4	5	6
7	8	

In the picture below, the numbers by each board tell you the order it was searched. (You can also see it at http://www.cse.buffalo.edu/~rapaport/572/S02/Graph1.GIF.)

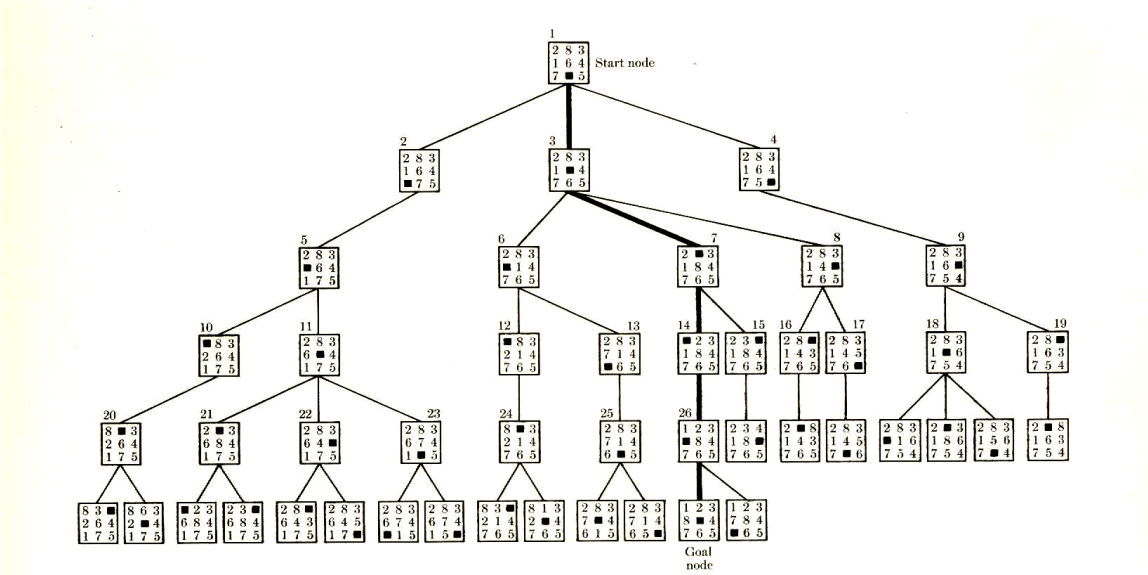

FIG. 3-2 *The tree produced by a breadth-first search.*

There aren't too many moves between the start state and the end state, but the tree is still pretty big. These search trees get really big when the board requires a lot of moves to solve. The path to the solution is marked with a thicker line. This search checks all moves, smart or dumb, until it finds the solution by brute force.

As we've said before, a better strategy is to sort the list of boards to check. Then the best boards you've found so far, in the shortest number of moves, are at the front of the list.

This makes you likelier to find the path to the solution. The tree below shows how much faster the search is for this example. The boards are still numbered in the order they're searched, but the black circle defines the depth + misplaced tile score, and you can find the right board in far fewer steps, just by scoring and sorting the boards in the open queue.

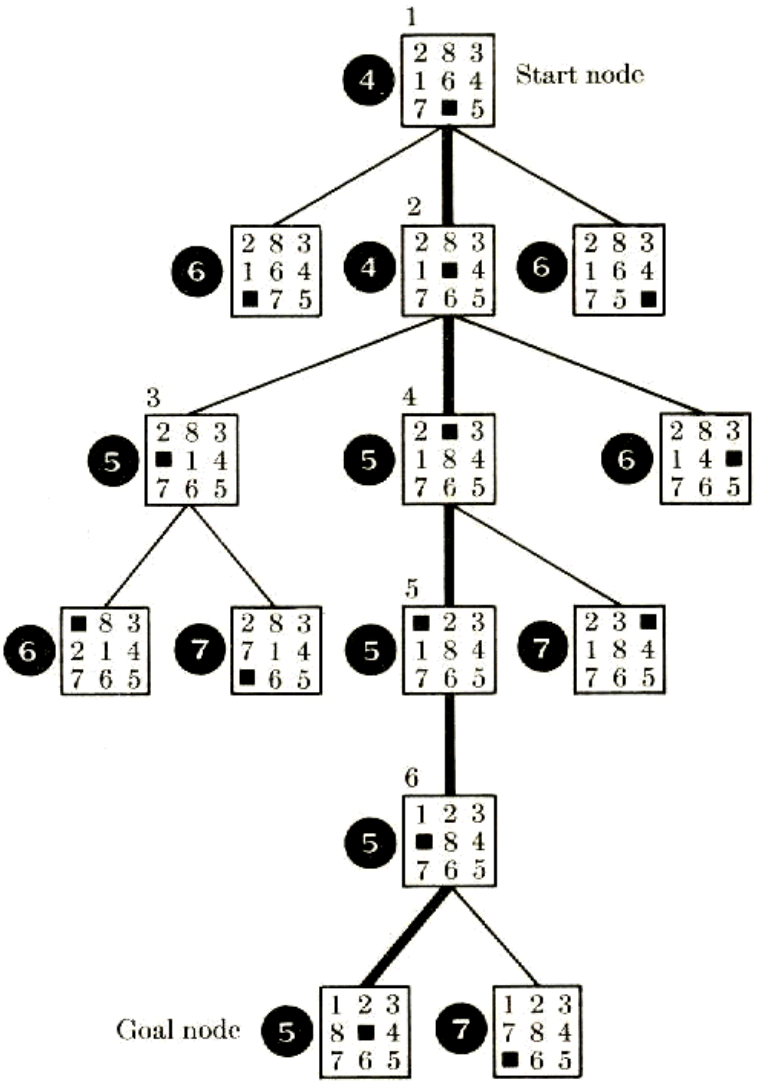

FIG. 3-6 *The tree produced by an ordered search.*

So first, go to the PuzzleBoard.java code and write the QuantMisplacedTiles function that scores each board. You can do this by writing a loop that looks at each tile and adds up the distance each tile is out of place for all the tiles 1-8, using the ManhattanDist function in the PTile library. This code is partly written. When you finally get the distance out of place added up for tiles 1-8, you should set the score to that distance (this is the score

variable in the PuzzleBoard class). Right now, the code sets all board scores to 2; you'll change this line to set the score variable to to whatever the true score is when you edit this function.

Next, you need to add a method to CleverSearch called EnqueueSorted. This method should take in two PuzzleBoard arrays, which may have different lengths, but are already each sorted. Your method will make a new array containing all the boards in sorted order. How? Let's look at a simple and partial sorting example:

```
char [] array1 = {'A', 'E', 'I', 'O', 'U'};
char [] array2 = {'C', 'F', 'J', 'M', 'X', 'Y', 'Z'};

char [] array3 = new char[array1.length + array2.length];
int i = 0;
int j = 0;
int k = 0;
while (k < array3.length)
{
        if (i < array1.length && j < array2.length)
        {
                if (array1[i] < array2[j])
                {
                        array3[k] = array1[i];
                        i = i + 1;
                        k = k + 1;
                }
                else
                {
                        array3[k] = array1[j];
                        j = j + 1;
                        k = k + 1;
                }
        }
        /* else, one of our arrays is used up; what should we do here? */
}
```

This is a start, but it's still a bit sketchy. What if we come to the end of array1 before array2? Then asking for the next element in array1 is most dangerous. So the code here still needs to be a bit smarter; try writing an else clause to deal with this case, and put it underneath the comment in the above code. If we've used up array1 but not array2, what should we do next? This code, plus that answer, gives you exactly the logic you'll use to build the sorted array of boards.

Finally, you'll need to plug in your new code. In the CleverSearch.Java file, find the SmartSearch method and locate the line

```
queue = EnqueueAtEnd(open, ExpandBoard(currentboard, closed));
```

Change this line to plug in your sorted array version:

```
queue = EnqueueSorted(open, ExpandBoard(currentboard, closed));
```

These two changes are all that are needed to make the search become smarter, so it finds the right answer much faster.