Report for Programming Problem 1 - 2048

Team: 2014230130

Student ID: 2014230130 Name: Ricardo Sintra Tavares

1. Algorithm description

The program starts by reading the input and iterating for each test case read. On each iteration a new object containing a matrix (array of arrays) is created, and filled with numbers (the board tiles).

Before calling the recursive function, the program checks if the board is already solved, if it can be solved in one move, and also if the board itself can be solved.
If none of the above are true, it starts calling the recursive function with 1 move already made, and setting the best possible number of plays as the maximum number of moves allowed for the board.

My recursive function performs one of the moves to the board (down, left, right, up). After moving, it checks if the board is solved and if true stops (setting the best var as the number of the move, if the number is lower than the current best), if it isn't solved it calls the recursive function again with the new board state after the move and increasing the move number by 1. This is repeated for each possible move.

To speed up the algorithm and discard unnecessary calls I apply several improvements:

If the number of the play is bigger than the best on memory, I don't make any move to the board. (a better solution has already been found)

On the move functions I set a variable to see if the move actually made anything, and if not, I don't call the recursive function again for that board state. This removes unnecessary calls if, for example, I try to move several times on one direction and have not changed the board.

On each recursive call the algorithm also checks if the board can actually be solved before calling again, this is achieved by checking if the sum of all tiles is a power of 2.

To check if the board was solved, I was iterating through the board after each move to see if the number of tiles was 1, but improved this approach by counting on the move itself, removing 2 nested for loops after each move.

I also tried adding each move to an array and sorting the array by the number of tiles each move produced, and calling the function in order of the least number of tiles, but couldn't get a positive result.

2. Data structures

My data structure is a struct declared as a global variable with an array of arrays set as the maximum possible board size, and a counter of the number of tiles the array has.

The array set to the maximum size (20) removes having to deal with memory allocation and freeing, by filling only the space needed for each test case, and replacing it on the case afterwards.

The counter is used to speed up checking if the board is solved.

Before each move a copy of this struct is made, the move is performed on the copy and if the conditions are met, the recursive function is called on the copy.

3. Correctness

I got 180 points as Wrong Answer. I think my solution is efficient in both memory allocation and execution time but I suspect I got a wrong answer because my algorithm fails if the number of moves is equal to the best moves possible to solve, I thought of this too late and didn't have time to correct it.

To fix it I suspect I have to change how I call the recursive function on the first call and change the verification of the play number being strictly lower than the best case.

4. Algorithm Analysis

On each move the board is iterated by a nested for loop (O(n^2)) followed by another for loop (O(n)), so each move has O(max(n^2, n) = O(n^2), with n being the boardsize.

After moving the board is checked for possibility of solving, again with a function that has time complexity of O(n^2).

On the recursive function, at worst it will call itself 4 times (one call for each move) so we have O(4^n).

The total complexity will be (4 * O(n^2) + 4 * O(n^2)) * O(4^n) = O(n^2) * O(4^n).

5. References

1. https://stackoverflow.com/questions/936687/how-do-i-declare-a-2d-array-in-c-using-new
2. https://stackoverflow.com/questions/27432964/tile-merging-algorithm-2048-game
3. EA slides.