Mandatory Project 1

Computationally hard problems

October 29, 2015

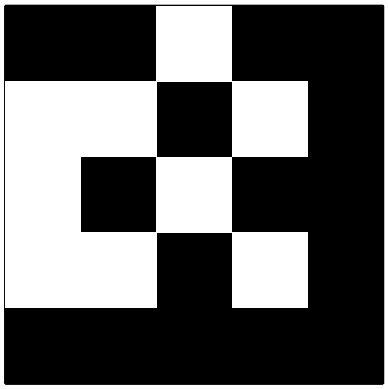
Troels Thomsen 152165
Rasmus Haarslev 152175

1 Madragon

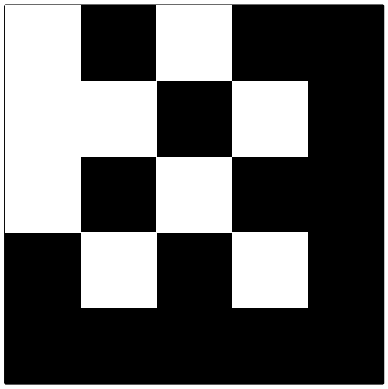
1.1 Determine whether the answer to the given Madragon instance is YES or NO

We take the following steps to shift the starting board to the goal board.

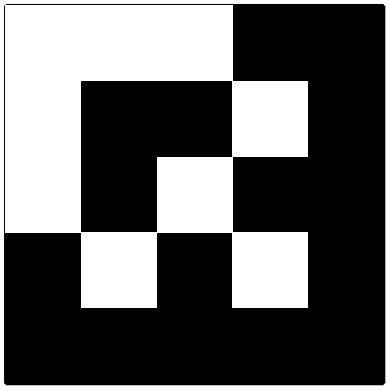
Starting board:



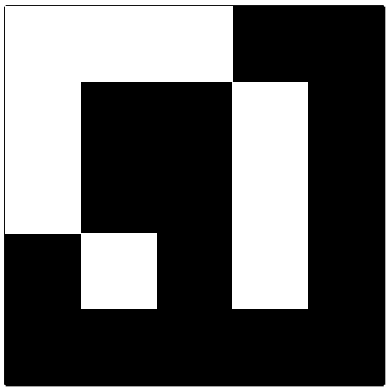
We move column 1 one tile up.



We column 2 two tiles down.



We move row 3 two tiles to the left.



And now we have made $k = 3$ operations and achieved the goal.

1.2 Show that Madragon is in NP

We specify the following randomized deterministic polynomial-bound algorithm A, which takes as input a problem instance X containing board states a and b and a random sequence of integers R.

1. 1a R is a random sequence of k integers chosen from an even distribution of numbers ranging from 1 to $(m+n)^2$.

   1b Every index in R corresponds a step. The value of each index corresponds to a move of either a column or a row. Since there exists $(m+n)^2$ unique moves, the value of R_i will determine which of these moves the algorithm performs at the ith step.

   1c After A has performed all i steps in R, A checks if the board has reached its goal state b and answers YES or NO accordingly.

2. 2a Since we choose R from an evenly distributed set of all possible combinations of size $\{1, \cdots, (m+n)^2\}^k = (m+n)^{2k}$, we know that if at least one R exists which solves $A(X, R) = YES$, we can choose this R with a probability of $\frac{1}{(m+n)^{2k}} > 0$.

   2b If there does not exist an R which solves $A(X, R) = YES$, then the algorithm A will never answer YES since the board cannot arrive at goal board b within k steps given any R.

3. In order to interpret R the algorithm A must generate all possible moves for the given board. This can be done in $O((m+n)^2)$ time. We can perform all steps in R in $O(k)$ time, and check the solution in $O(1)$ time. Thus the algorithm is bound by $O((m+n)^2)$.

1.3 Show that Madragon is NP-complete

1.4 Describe the solution for the optimization version of the algorithm

Our solution assumes, that for any given board A, there exists a set of moves R_0, such that A may be transformed into any given board B, where A and B consists of the same number of black and white tiles.

Our optimization algorithm A_o generates the set of possible moves $S = \{s_0, s_1, ..., s_n\}$ and the set $R = \{R_0, R_1, ..., R_m\}$, containing every combination of $k - length$ move-sets from S.

We run all of the move-sets in R against board A, and if one of the sets in R transform board A into goal board B, then the algorithm returns the move-set R_0, else $false$.

We run this process iteratively starting with $k = 1$ going up to $k = max_k$ where max_k is the k given in the .mad file.

1.5 Prove the worst-case running time of your algorithm

In the worst case, A_o runs through all of the generated sets in R. Given that R contains all possible permutations of k-length sets we have $|R| = |S|^k = ((m+n)^{\frac{(m-1)+(n-1)}{2}})^k$ since $|S| = ((m+n)^{\frac{(m-1)+(n-1)}{2}})^k$. We can check each of these sets in $O(k)$ time since each set is of length k and each move can be performed in $O(1)$ constant time.

This gives A_o a worst-case running time of $T(A_o) = O(|S|^k \times k) = O(|S|^k)$.

1.6 Implement the algorithm for Madragon

Out implementation can be run by running the file madragon:

```
./madragon -f <path-to-file.mad>
```

Our implementation returns an array of the following format:

```
[k, [
    [row / column, rotations], ..., ...]
]
```

Where k is the k for which a solution was found, and where $row/column$ is an integer representing which row or column to rotate. If the number is less than the number of

rows, we rotate the 0-indexed row in the number of rows on the board. If the number is greater than or equal to the number of rows, we rotate the 0-indexed column in the number of columns on the board.

rotations correspond to the number of times we rotate the given row or column. We always rotate from right to left or bottom to top.

For the file $arecibo - 1.mad$ the result would be:

```
[1, [[8, 1]]]
```

Indicating that we can solve the board with a 1-length set of moves, and a move-set consisting of a single move where we rotating column 4 one step.