107062381 楊孝偉

HW1 Report

Pseudo code

The basic idea in solving this problem is to use divide-and-conquer algorithm. The following is the pseudo code for my program.

1. Base case : if the square is of size 2 x 2, simply fill the remaining unfilled cells.
2. Else if the square is bigger than 2 x 2 :
   a. Find the “filled” cell, whether its in the 1st, 2nd, 3rd or 4th quadrants
   b. Place an L shaped tile at the center of the grid such that it doesn’t cover the sub-square that has the filled cell. Suppose the filled cell found in (a) is in 1st quadrant, then we place the L shaped tile at the center such that it will cover 2nd, 3rd and 4th quadrants.
   c. Solve the 4 quadrants recursively.

In the following, I will explain the code in more details.

For the main function, it will first take the number of testcases. Then for each test case, we first reset the “cnt” counter to 0, this counter will be used in the future for marking the “L” tile on the grid. Then, it will take the value for k, which will be used to define the size_of_grid by 2^k using the pow function. Then we need to reset all elements in the array to 0 as to ease us to identify which tiles haven’t been filled in the future. Then we take a and b to mark the a^{th} row and b^{th} column in the array to -1. Then we start entering the function tile(), and it will take size_of_grid as one of its parameters. The other parameters are used to indicate the starting point of the whole grid, I will explain this as we go further, the initial starting points are set to 0^{th} row and 0^{th} column of the whole grid first.

Entering the tile function, initially, int x and int y are both set to 0 as this is the first iteration of the tile function. For this first iteration, it will check first whether n is of size 2 or not, if n is size 2, then the grid will only be of 2 by 2 size and this is the base case. If this condition is met, then we increase the “cnt” counter that we mentioned previously in the main function, then we simply look for which tiles in the 2 by 2 tiles that are equal to 0 and mark it with cnt (this is why I set all of the elements value to 0 previously to ease this finding of unmarked tiles). Else if n isn’t of size 2 by 2, then we need to find the “marked” tiles, whether it’s in the 1st, 2nd, 3rd or 4th quadrant. Finding the whole needs to start from x and y, because later on, x and y will be set to the first grid of the sub quadrants when entering the recursion. This is very important as my first bug when writing this code was because of this mistake that I made.

Then simply look for which quadrant is the “marked” tile in. we have to place the L shaped tile in the quadrants that don’t contain the “marked” tile. If the marked tile is in 1st quadrant, then the L shaped tile should be placed in the 2nd, 3rd and 4th quadrants of the grid. This is done by the place function. After placing the L shaped tile, then do the recursions for the four sub-quadrants. When checking which quadrants has the marked tile, we need to add x and y to the n/2, this is to ensure that it it start counting from the supposed starting point. This was also the mistake that I made when creating this program.

107062381 楊孝偉

Time Complexity

The time complexity of my code is O(n^{2l}lgn). This can be proven with master theorem. The recurrence relationship is :

$T(n) = 4T(n/2) + cn^2$, $n > 2$

$T(n) = cn^2$, $n = 2$ (base case)

T(n/4) is because of the recursion process for the 4 quadrants, and the cn^2 is when finding the "filled" tile location.

$Log_2 4$ is equal to 2. It is equal to the power of n^2, which is 2, therefore it matches the case 2 of master theorem, where :

$T(n) = aT(n/b) + f(n)$ where $a >= 1$ and $b > 1$

There are following three cases:
1. If $f(n) = \Theta(n^c)$ where $c < Log_b a$ then $T(n) = \Theta(n^{Log_b a})$
2. If $f(n) = \Theta(n^c)$ where $c = Log_b a$ then $T(n) = \Theta(n^c Log\ n)$
3.If $f(n) = \Theta(n^c)$ where $c > Log_b a$ then $T(n) = \Theta(f(n))$