THE DEFECTIVE CHESSBOARD PROBLEM

A MINOR PROJECT REPORT

Submitted by

SWARALI UTEKAR **RA2011003010377**

Under the guidance of

Dr.N.ARUNACHALAM **Assistant Professor**

In a partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

of

FACULTY OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
Deemed to be University u/s 3 of UGC Act, 1956

S.R.M. Nagar, Kattankulathur, Chengalpattu District

JUNE 2022

ABSTRACT

The Defective Chessboard problem, also known as the Tiling Problem is an interesting problem. It is typically solved with a “divide and conquer” approach. The algorithm has a time complexity of $O(n^2)$.As mentioned earlier, a divide-and-conquer (DAC) technique is used to solve the problem. DAC entails splitting a larger problem into sub-problems, ensuring that each sub-problem is an exact copy of the larger one, albeit smaller.

CONTRIBUTION TABLE :

SWARALI UTEKAR	RA2011003010377

TABLE OF CONTENT

Problem Definition

Given a n × n board where n is of form 2^k where k >= 1 (Basically, n is a power of 2 with minimum value as 2). The board has one missing square). Fill the board using triominoes. A triomino is an L-shaped tile is a 2 × 2 block with one cell of size 1×1 missing.

Problem Explanation

Given Conditions:

1. We have a chessboard of size n × n, where n= 2^k, for some k>=1.
2. Exactly one square is defective in the chessboard, i.e., exactly one square is missing
3. The tiles(triominoes) are in L – shape, i.e., 3 squares.

Objective:

Cover all the chessboard with L-shape tiles(triominoes), except the defective square.

Design Technique Used

DIVIDE AND CONQUER ALGORITHMIC TECHNIQUE

Divide and Conquer is an algorithm design paradigm. A divide-and-conquer algorithm recursively breaks down a large problem into two or more smaller sub-problems of the same or related type, until these problems become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a solution to the original problem.

Algorithm for the problem

// n is size of given square, p is location of missing cell

Chessboard(int n, Point p)

1) Base case: n = 2, A 2 x 2 square with one cell missing is nothing but a tile and can be filled with a single tile.
2) Place a L shaped tile at the center such that it does not cover the n/2 * n/2 subsquare that has a missing square. **Now all four subsquares of size n/2 x n/2 have a missing cell** (a cell that doesn't need to be filled).
3) Solve the problem recursively for following four. Let p1, p2, p3 and p4 be positions of the 4 missing cells in 4 squares.
   a) Chessboard(n/2, p1)
   b) Chessboard(n/2, p2)
   c) Chessboard(n/2, p3)
   d) Chessboard(n/2, p3)

Sample Input/Output:

Input: size = 2 and mark coordinates = (0, 0)

Output:

-1 1

1 1

Coordinate (0, 0) is marked. So, no tile is there. In the remaining three positions,

a tile is placed with its number as 1.

Input: size = 4 and mark coordinates = (0, 0)

Output:

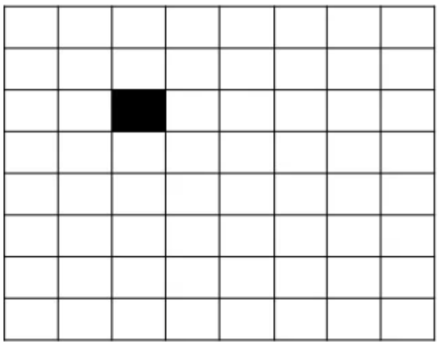
-1	3	2	2
3	3	1	2
4	1	1	5
4	4	5	5

Explanation of Algorithm with an Example

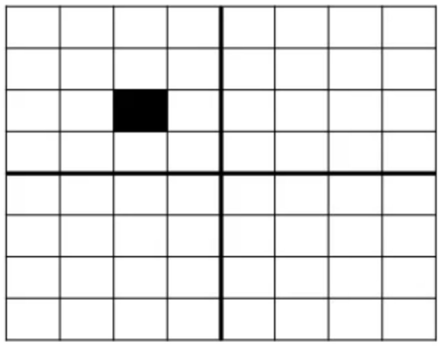
- A chessboard is an n x n grid, where n is a power of 2.
- A defective chessboard is a chessboard that has one unavailable (defective) position.
- A triomino is an L shaped object that can cover three squares of a chessboard. A triomino has four orientations.
- Place $(n^2 - 1)/3$ triominoes on an n x n defective chessboard so that all $n^2 - 1$ non defective positions are covered.
- For an 8 x 8 chessboard, divide the chessboard into four smaller chessboards. (4 x 4)
- One of these is a defective 4 x 4 chessboard.
- Make the other three 4 x 4 chessboards defective by placing a triomino at their common corner.
- Recursively tile the four defective 4 x 4 chessboards.

8X8 DEFECTIVE CHESS BOARD

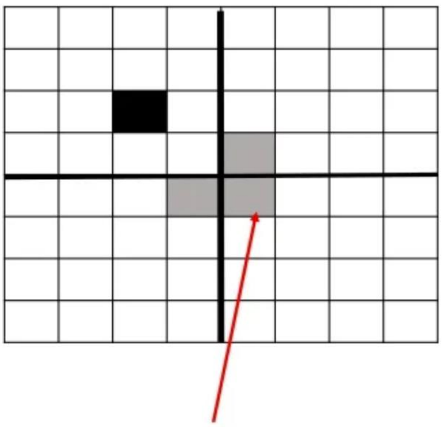
Step-1 One of the cell is defective



Step- 2 We divide the chess board into equal sub half's.

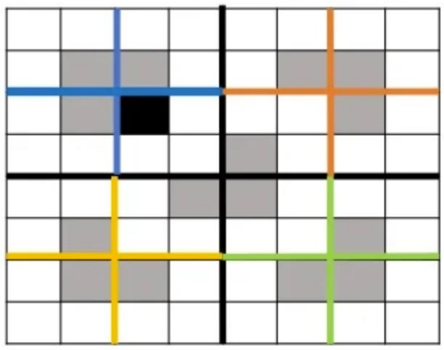


Step- 3 Trick to cover the chess board with tiles



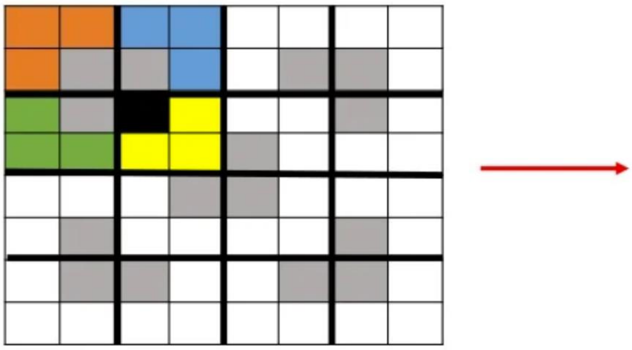
Creation of defective box

Step -4 Again creation of defective boxes as we divide the chess board

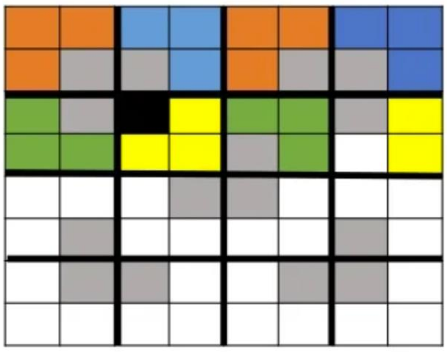


DIVISION OF PROBLEM INTO SUB PROBLEM

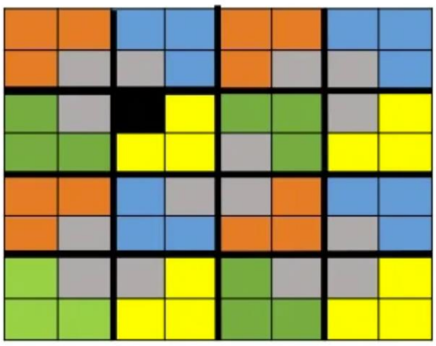
Step-5 As we have finally divided the problem into 2x2 board we will put the tiles.



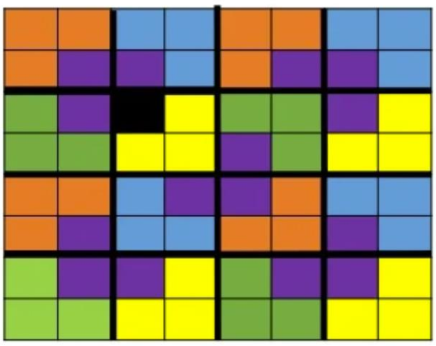
Step-6 The procedure will continue until all the sub board are covered with the tiles.



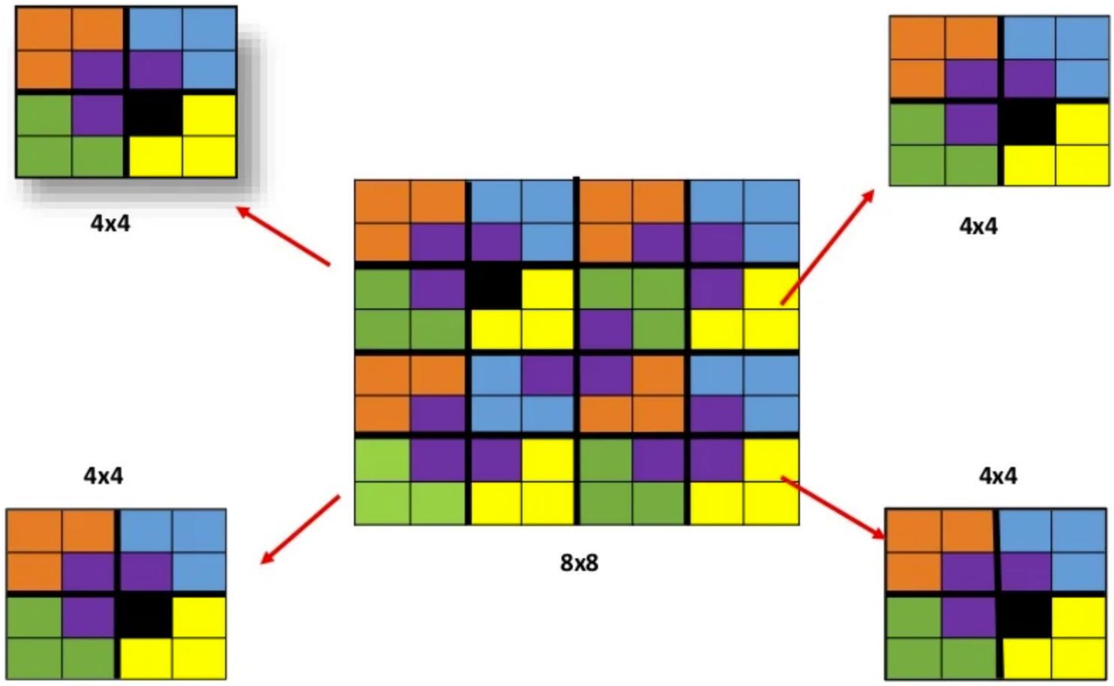
Step-7 The final chess board covered with all the titles and only left with the defectives which we created.

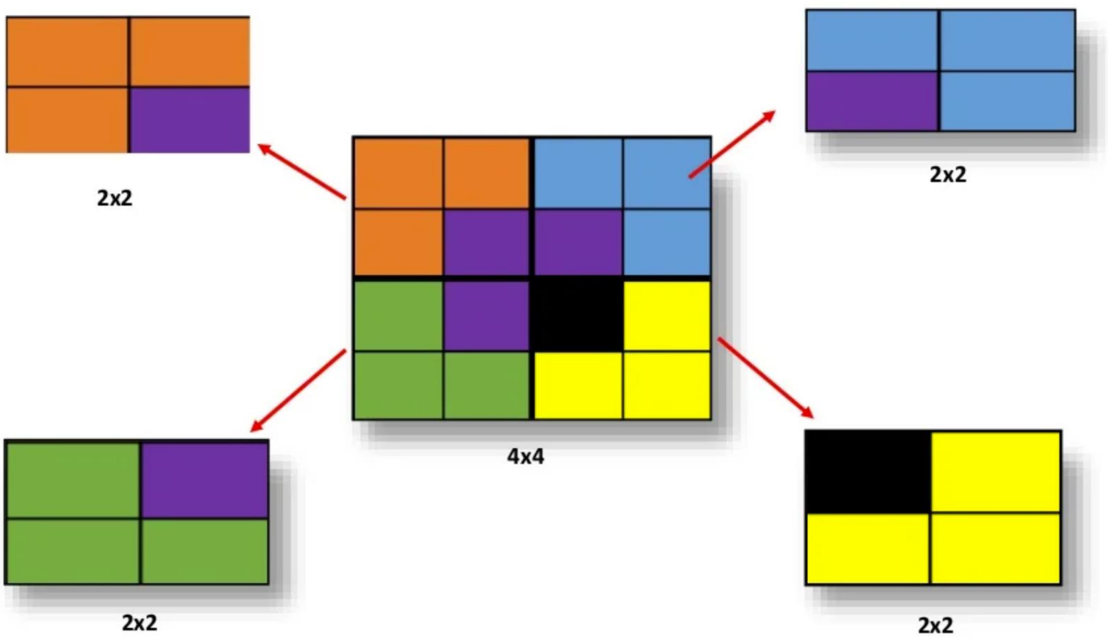


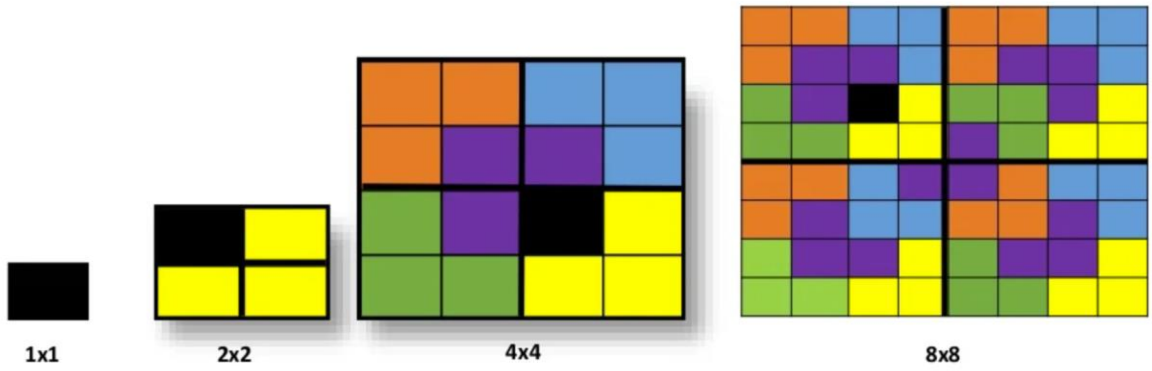
Step-7 Here we will cover the defectives which we have created as in the last, there should be only one defective left.



COMBINIG OF ALL SUB PROBLEMS


4x4
4x4
4x4
4x4
8x8


2x2
2x2
4x4
2x2
2x2



<u>Complexity Analysis</u>

Recurrence relation for above recursive algorithm can be written as below:

T(n) = 4T(n/2) + C , C is a constant.

The above recursion can be solved below using Master Theorem:

$T(n) = a\, T(n/b) + \Theta\, (n^k \log^p n)$, $a>=1$, $b>1$, $k>=0$ and p is a real number.

Here, a=4, b=2, k=0, p=0

$\log_b a \Rightarrow \log_2 4 \Rightarrow 2$

Since $\log_b a > k$ as $2 > 0$,

Case l: If $\log_b a > k$,

then $T(n) = \Theta\, (n^{\log_b a}) \Rightarrow \Theta\, (n^2)$

$\mathbf{T(n) = O\, (n^2)}$

Hence, the **Time Complexity** for this problem is $\mathbf{O(n^2)}$.

Implementation

Code:

```
// C++ program defective chessboard
#include <bits/stdc++.h>
using namespace std;

int size_of_grid, b, a, cnt = 0;
int arr[128][128];

// Placing tile at the given coordinates
void place(int x1, int y1, int x2,
        int y2, int x3, int y3)
{
    cnt++;
    arr[x1][y1] = cnt;
    arr[x2][y2] = cnt;
    arr[x3][y3] = cnt;
}
// Quadrant names
// 1 2
// 3 4

// Function based on divide and conquer
int chessboard(int n, int x, int y)
{
    int r, c;
    if (n == 2) {
        cnt++;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (arr[x + i][y + j] == 0) {
                    arr[x + i][y + j] = cnt;
                }
            }
        }
        return 0;
    }
    // finding hole location
    for (int i = x; i < x + n; i++) {
        for (int j = y; j < y + n; j++) {
            if (arr[i][j] != 0)
                r = i, c = j;
        }
    }

    // If missing Tile is 1st quadrant
```

```
    if (r < x + n / 2 && c < y + n / 2)
        place(x + n / 2, y + (n / 2) - 1, x + n / 2,
            y + n / 2, x + n / 2 - 1, y + n / 2);

    // If missing Tile is in 3rd quadrant
    else if (r >= x + n / 2 && c < y + n / 2)
        place(x + (n / 2) - 1, y + (n / 2), x + (n / 2),
            y + n / 2, x + (n / 2) - 1, y + (n / 2) - 1);

    // If missing Tile is in 2nd quadrant
    else if (r < x + n / 2 && c >= y + n / 2)
        place(x + n / 2, y + (n / 2) - 1, x + n / 2,
            y + n / 2, x + n / 2 - 1, y + n / 2 - 1);

    // If missing Tile is in 4th quadrant
    else if (r >= x + n / 2 && c >= y + n / 2)
        place(x + (n / 2) - 1, y + (n / 2), x + (n / 2),
            y + (n / 2) - 1, x + (n / 2) - 1,
            y + (n / 2) - 1);

    // dividing it again in 4 quadrants
    chessboard(n / 2, x, y + n / 2);
    chessboard(n / 2, x, y);
    chessboard(n / 2, x + n / 2, y);
    chessboard(n / 2, x + n / 2, y + n / 2);

    return 0;
}
// Driver program to test above function
int main()
{
    // input size of chessboard
    cout<<"Enter the size of the chessboard: ";
    cin>>size_of_grid;
    memset(arr, 0, sizeof(arr));
    // Coordinates which will be marked
    a = 0, b = 0;
    // Here tile can not be placed
    arr[a][b] = -1;
    chessboard(size_of_grid, 0, 0);
    // The grid is
    for (int i = 0; i < size_of_grid; i++) {
        for (int j = 0; j < size_of_grid; j++)
            cout << arr[i][j] << " \t";
        cout << "\n";
    }
}
```

Input/Output:

```
Enter the size of the chessboard: 2
-1      1
1       1
```

```
Enter the size of the chessboard: 4
-1      3       2       2
3       3       1       2
4       1       1       5
4       4       5       5
```

```
Enter the size of the chessboard: 8
-1      9       8       8       4       4       3       3
9       9       7       8       4       2       2       3
10      7       7       11      5       5       2       6
10      10      11      11      1       5       6       6
14      14      13      1       1       19      18      18
14      12      13      13      19      19      17      18
15      12      12      16      20      17      17      21
15      15      16      16      20      20      21      21
```

Conclusion

This report explains how a defective or missing square in an n x n chessboard can be found using the concept of Divide and Conquer Algorithmic Technique. It is a real time application for this approach.

References

[1]. Introduction to the Design and Analysis of Algorithms, by Anany Levitin.

[2]. https://www.geeksforgeeks.org/

[3]. https://leetcode.com/