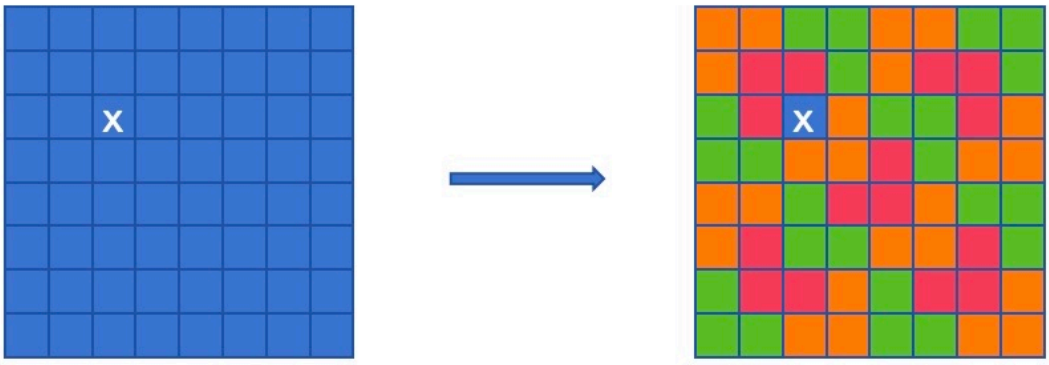
CS208 Lab7 Practice

12312110 李轩然

DDL: May.4

Chessboard Coverage Problem

Given a $2^n \times 2^n$ chessboard, exactly one square is missing. The entire chessboard should be covered with L-shaped dominoes (composed of three squares, shaped like an L), without overlapping or covering any missing squares.



Analysis

When $n = 1$, a L-shaped domino can easily fill the board with a missing square.

Then divide the $2^n \times 2^n$ board into four $2^{n-1} \times 2^{n-1}$ sub-boards, and we have done the case of $2^{n-1} \times 2^{n-1}$ sub-board with a missing square. Put a domino in the center, so that it fills a square of three of the sub-board. Then for these three sub-board, the question becomes filling them ($2^{n-1} \times 2^{n-1}$) with a missing square. That's what we have done before. And for the left sub-board, using the same method and it will have a missing board as well.

The $n = 1$ case takes $O(1)$ time, and every time we divide the problem into four sub-problems, so for time complexity, the recurrence relation is $T(n) = 4T(n-1) + O(1)$. Thus $T(n)$ is $O(4^n)$, which has the same size of total number of squares.

C++ Code

```cpp
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

int domino_num = 0;
```

```
void printBoard(const vector<vector<int>>& board) {
    for (const auto& row : board)
    {
        for (int val : row)
            if (val == -1) cout << "X\t"; else cout << val << "\t";
        cout << "\n";
    }
}

void placeDomino(vector<vector<int>>& board, int row1, int col1, int row2, int col2,
int row3, int col3) {
    domino_num ++;
    board[row1][col1] = domino_num;
    board[row2][col2] = domino_num;
    board[row3][col3] = domino_num;
}

void coverBoard(vector<vector<int>>& board, int size, int top_row, int left_col, int
missing_row, int missing_col) {
    if (size == 2) {
        domino_num ++;
        for (int i = top_row; i < top_row + 2; i++)
            for (int j = left_col; j < left_col + 2; j++)
                if (i != missing_row or j != missing_col) board[i][j] = domino_num;
        return;
    }

    int half = size / 2;

    int quadrant;
    if (missing_row < top_row + half and missing_col < left_col + half) {
        quadrant = 1;
    } else if (missing_row < top_row + half and missing_col >= left_col + half) {
        quadrant = 2;
    } else if (missing_row >= top_row + half and missing_col < left_col + half) {
        quadrant = 3;
    } else quadrant = 4;

    if (quadrant != 1) {
        board[top_row + half - 1][left_col + half - 1] = domino_num;
    }
    if (quadrant != 2) {
        board[top_row + half - 1][left_col + half] = domino_num;
    }
    if (quadrant != 3) {
        board[top_row + half][left_col + half - 1] = domino_num;
    }
    if (quadrant != 4) {
        board[top_row + half][left_col + half] = domino_num;
    }

    domino_num ++;

    if (quadrant == 1) {
```

```
        coverBoard(board, half, top_row, left_col, missing_row, missing_col);
    } else {
        coverBoard(board, half, top_row, left_col, top_row + half - 1, left_col +
half - 1);
    }

    if (quadrant == 2) {
        coverBoard(board, half, top_row, left_col + half, missing_row, missing_col);
    } else {
        coverBoard(board, half, top_row, left_col + half, top_row + half - 1,
left_col + half);
    }

    if (quadrant == 3) {
        coverBoard(board, half, top_row + half, left_col, missing_row, missing_col);
    } else {
        coverBoard(board, half, top_row + half, left_col, top_row + half, left_col +
half - 1);
    }

    if (quadrant == 4) {
        coverBoard(board, half, top_row + half, left_col + half, missing_row,
missing_col);
    } else {
        coverBoard(board, half, top_row + half, left_col + half, top_row + half,
left_col + half);
    }
}

int main()
{
    int n;
    cin >> n;

    int size = pow(2, n);
    vector<vector<int>> board(size, vector<int>(size, 0));
    int missing_row, missing_col;
    cin >> missing_row >> missing_col;

    board[missing_row][missing_col] = -1;
    coverBoard(board, size, 0, 0, missing_row, missing_col);
    printBoard(board);

    return 0;
}
```