Input

- n: The size of the chessboard (in this case, 12 for a 12x12 board).
- (r1, c1) and (r2, c2): Coordinates of two cells that are removed from the board

Variable Representation:

- Each cell (i, j)on the board is represented by four variables:
- Xi,j,1 (a): Indicates the cell is covered by the left side of a horizontally placed domino.
- Xi,j,-1 (b): Indicates the cell is covered by the right side of a horizontally placed domino.
- Xi,j,2 (c): Indicates the cell is covered by the top side of a vertically placed domino.
- Xi,j,-2 (d): Indicates the cell is covered by the bottom side of a vertically placed domino.
- The variable counter keeps track of these four variables for each cell.

Single Domino Coverage:

Each cell must be covered by exactly one domino. The code enforces this by creating clauses:

- {a, b, c, d} ensures that one of these options must be true (at least one direction covers the cell).
- {-a, -b}, {-a, -c}, {-a, -d}, {-b, -c}, {-b, -d}, and {-c, -d} ensure that no two directions simultaneously cover the cell

Neighbor to the Right :

- newcolumn = column + 1: Sets the potential right neighbor's column.

Boundary Check:

- if(newcolumn > n || newrow > n || (newrow == r1 && newcolumn == c1) || (newrow == r2 && newcolumn == c2)): Checks if this right neighbor is outside the board bounds or one of the removed cells.
- If true, clauses.push_back({-a}) adds a clause {-a}, indicating that if cell (i, j) has a (left side of a horizontal domino) set, it cannot find a valid right neighbor to pair with.

Valid Right Neighbor:

- If a valid neighbor exists, clauses.push_back({-a, a+5}) links a to its neighboring variable a + 5, enforcing that if the current cell is covered by the left side of a horizontal domino, the right neighbor must be covered by the right side of the same domino.

Neighbor to the Left :

- Set Left Neighbor:
- newcolumn = column - 1 sets the potential left neighbor's column index by moving one cell to the left.

Boundary Check:

- The condition if (newcolumn < 1 || newrow < 1 || (newrow == r1 && newcolumn == c1) || (newrow == r2 && newcolumn == c2)) checks if this left neighbor is outside the grid bounds (i.e., newcolumn < 1) or is one of the removed cells at (r1, c1) or (r2, c2).
- If true, clauses.push_back({-b}) adds a clause {-b}, indicating that if cell (i, j) has b (right side of a horizontal domino) set, it cannot find a valid left neighbor to pair with.

Valid Left Neighbor:

- If a valid left neighbor exists, clauses.push_back({-b, b-5}) links b (right side of the domino) to

the neighboring variable b - 5, representing the left side of the horizontal domino.

Neighbor Below :

- newrow = row + 1: Sets the potential lower neighbor's row

Boundary Check:

- if(newrow > n || (newrow == r1 && newcolumn == c1) || (newrow == r2 && newcolumn == c2)): Ensures the potential lower neighbor is within bounds and not a removed cell.
- If true, clauses.push_back({-c}) adds a clause {-c}, meaning if cell (i, j) has c (top side of a vertical domino) set, there is no valid bottom neighbor to pair with.

Valid Lower Neighbor:

- If a valid lower neighbor exists, clauses.push_back({-c, c + 1 + tobeadded}) links c to the adjusted neighbor variable, enforcing the pairing.

Calculating tobeadded

- Since each row contains n cells, and each cell has four variables:

One row of cells has n * 4 variables.

Thus:

tobeadded = n * 4 ensures that we jump to the correct variable index for the row below.

Adjustments to tobeadded for Removed Cells

- If there are removed cells in certain positions (relative to the current cell), they might affect the continuity of variable indexing. For instance:
- If a cell is removed to the left or above the current cell, it affects how we count variables in the row below.
- To correct for this, the code decreases tobeadded by 4 whenever such a removed cell exists.

Neighbor Above :

newrow = row - 1: Sets the potential upper neighbor.

Boundary Check:

- If the upper neighbor is out of bounds or a removed cell, clauses.push_back({-d}) adds a clause {-d}, indicating that the current cell cannot be covered by the bottom side of a vertical domino.

Valid Upper Neighbor:

- If a valid neighbor exists, clauses.push_back({-d, d - 1 - tobesubtracted}) links d to the corresponding neighbor, ensuring the domino covers both cells.

Calculating tobesubtracted

- To ensure we reach the correct variable index of the upper neighbor, tobesubtracted is set to:
- tobesubtracted = n * 4, which shifts the variable index back by one row.

Adjustments to tobesubtracted for Removed Cells:

- If removed cells are near the current cell, they can affect variable indexing for the row above. To address this, tobesubtracted is decreased by 4 (the number of variables per cell) when:
- A cell is removed to the right or below the current cell.
- A cell is removed to the left or above the cell directly above.