Coloring Border in C++

```
#include <iostream>
#include <vector>

using namespace std;

vector<vector<int>> dirs = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}};

void dfs(vector<vector<int>>& grid, int row, int col, int
clr) {
    grid[row][col] = -clr;
    int count = 0;

    for (auto dir : dirs) {
        int rowdash = row + dir[0];
        int coldash = col + dir[1];

        if (rowdash < 0 || coldash < 0 || rowdash >=
grid.size() || coldash >= grid[0].size() ||
abs(grid[rowdash][coldash]) != clr) {
            continue;
        }

        count++;

        if (grid[rowdash][coldash] == clr) {
            dfs(grid, rowdash, coldash, clr);
        }
    }

    if (count == 4) {
        grid[row][col] = clr;
    }
}

void coloring_border(vector<vector<int>>& grid, int row,
int col, int color) {
    dfs(grid, row, col, grid[row][col]);

    for (int i = 0; i < grid.size(); i++) {
        for (int j = 0; j < grid[0].size(); j++) {
            if (grid[i][j] < 0) {
                grid[i][j] = color;
            }
        }
    }
}

int main() {
    // Hardcoded input
    int m = 4;
    int n = 4;
    vector<vector<int>> arr = {
        {2, 1, 3, 4},
        {1, 2, 2, 2},
        {3, 2, 2, 2},
        {1, 2, 2, 2}
    };
    int row = 1;
    int col = 1;
    int color = 3;
```

Step-by-Step Dry Run:

Step 1: Call to coloring_border

- **Initial call:** coloring_border(arr, row=1, col=1, color=3).
- Call dfs(grid, row=1, col=1, clr=2) to mark the connected component and determine the border cells.

Step 2: DFS Traversal

The function dfs will:

1. Mark cells in the connected component as -2 (negate the value).
2. Identify border cells and keep them marked as -2. Interior cells are restored to clr (2) if they are surrounded by four other cells of the same value.

Traversal Process:

1. Start DFS at (1, 1):
    - Mark grid[1][1] = -2.
    - Neighboring cells:
        - (1, 2) → Proceed (same value 2).
        - (2, 1) → Proceed (same value 2).
        - Other directions lead to invalid cells or cells with a different value.
    - **Connected cells count so far:** 2.
2. Move to (1, 2):
    - Mark grid[1][2] = -2.
    - Neighboring cells:
        - (1, 3) → Proceed (same value 2).
        - (2, 2) → Proceed (same value 2).
    - **Connected cells count so far:** 4.
3. Move to (1, 3):
    - Mark grid[1][3] = -2.
    - Neighboring cells:
        - (2, 3) → Proceed (same value 2).
4. Continue this process until all connected cells are visited and marked as -2. For the input grid, all 2 values connected to (1, 1) are part of the connected component.

```
    coloring_border(arr, row, col, color);

    // Print the modified grid
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            cout << arr[i][j] << "\t";
        }
        cout << endl;
    }

    return 0;
}
```

- **Marked Grid after DFS:**

  Copy code
  2 1 3 4
  1 -2 -2 -2
  3 -2 -2 -2
  1 -2 -2 -2

Step 3: Check for Border Cells

The DFS function identifies border cells:

- A cell is on the border if it:
  - Is not surrounded on all four sides by other cells of the same value.
- Border cells remain marked as -2.

Step 4: Update Colors

- Traverse the grid:
  - Any cell marked as -2 is changed to the new color (3).
  - Interior cells (those surrounded by the same value) are reverted to clr (2).
- **Final Grid:**

  2 1 3 4
  1 3 3 3
  3 3 3 3
  1 3 3 3

Output:-

```
2    1    3    4
1    3    3    3
3    3    2    3
1    3    3    3
```