Largest submatrix C++

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

// Define the maximum size for the grid (you can adjust this as needed)
const int MAX_ROWS = 100;
const int MAX_COLS = 100;

// Function to find the largest square submatrix
int largestSquareSubmatrix(const int arr[MAX_ROWS][MAX_COLS], int rows, int cols) {
    int dp[MAX_ROWS][MAX_COLS] = {0}; // DP table
    int largestSide = 0;

    // Fill the dp array
    for (int i = rows - 1; i >= 0; i--) {
        for (int j = cols - 1; j >= 0; j--) {
            if (i == rows - 1 || j == cols - 1) {
                dp[i][j] = arr[i][j];
            } else {
                if (arr[i][j] == 0) {
                    dp[i][j] = 0;
                } else {
                    int minSide = min(dp[i][j + 1], min(dp[i + 1][j], dp[i + 1][j + 1]));
                    dp[i][j] = minSide + 1;
                }
            }
            if (dp[i][j] > largestSide) {
                largestSide = dp[i][j];
            }
        }
    }

    return largestSide; // Return the side length of the largest square submatrix
}

int main() {
    // Define the array and its dimensions
    const int arr[MAX_ROWS][MAX_COLS] = {
        {0, 1, 0, 1, 0, 1},
        {1, 0, 1, 0, 1, 0},
        {0, 1, 1, 1, 1, 0},
        {0, 0, 1, 1, 1, 0},
        {1, 1, 1, 1, 1, 1}
    };
    int rows = 5;
    int cols = 6;

    cout << largestSquareSubmatrix(arr, rows, cols) << endl;

    return 0;
}
```

Dry Run of the Program

Let's break down how the program works with the input grid:

Input:

- The input grid arr[MAX_ROWS][MAX_COLS] is:

Copy code
```
0 1 0 1 0 1
1 0 1 0 1 0
0 1 1 1 1 0
0 0 1 1 1 0
1 1 1 1 1 1
```

- The number of rows (rows) = 5
- The number of columns (cols) = 6

Initializations:

- dp[MAX_ROWS][MAX_COLS] is initialized to 0.
- largestSide = 0, which will keep track of the largest side of the square submatrix found.

The DP table (dp[i][j]) will store the size of the largest square submatrix whose bottom-right corner is at position (i, j).

Process:

We start iterating from the bottom-right corner of the matrix (i = rows - 1, j = cols - 1) and move upwards and to the left.

Iteration details:

1. **For i = 4, j = 5**:
    - arr[4][5] = 1
    - Since it's the last row or column (i == rows - 1 or j == cols - 1), dp[4][5] = arr[4][5] = 1.
    - largestSide = max(largestSide, dp[4][5]) = max(0, 1) = 1.
2. **For i = 4, j = 4**:
    - arr[4][4] = 1
    - Since it is the last row or column, dp[4][4] = arr[4][4] = 1.
    - largestSide = max(largestSide, dp[4][4]) = max(1, 1) = 1.
3. **For i = 4, j = 3**:
    - arr[4][3] = 1
    - Since it's the last row or column,

	dp[4][3] = arr[4][3] = 1. - largestSide = max(largestSide, dp[4][3]) = max(1, 1) = 1. 4. **For i = 4, j = 2**: - arr[4][2] = 1 - Since it's the last row or column, dp[4][2] = arr[4][2] = 1. - largestSide = max(largestSide, dp[4][2]) = max(1, 1) = 1. 5. **For i = 4, j = 1**: - arr[4][1] = 1 - Since it's the last row or column, dp[4][1] = arr[4][1] = 1. - largestSide = max(largestSide, dp[4][1]) = max(1, 1) = 1. 6. **For i = 4, j = 0**: - arr[4][0] = 1 - Since it's the last row or column, dp[4][0] = arr[4][0] = 1. - largestSide = max(largestSide, dp[4][0]) = max(1, 1) = 1. 7. **For i = 3, j = 5**: - arr[3][5] = 0 - Since arr[3][5] == 0, dp[3][5] = 0. 8. **For i = 3, j = 4**: - arr[3][4] = 1 - dp[3][4] = min(dp[3][5], min(dp[4][4], dp[4][5])) + 1 = min(0, min(1, 1)) + 1 = 1. - largestSide = max(largestSide, dp[3][4]) = max(1, 1) = 1. 9. **For i = 3, j = 3**: - arr[3][3] = 1 - dp[3][3] = min(dp[3][4], min(dp[4][3], dp[4][4])) + 1 = min(1, min(1, 1)) + 1 = 2. - largestSide = max(largestSide, dp[3][3]) = max(1, 2) = 2. 10. **For i = 3, j = 2**: - arr[3][2] = 1 - dp[3][2] = min(dp[3][3], min(dp[4][2], dp[4][3])) + 1 = min(2, min(1, 1)) + 1 = 2. - largestSide = max(largestSide, dp[3][2]) = max(2, 2) = 2. 11. **For i = 3, j = 1**: - arr[3][1] = 0 - Since arr[3][1] == 0, dp[3][1] = 0. 12. **For i = 3, j = 0**: - arr[3][0] = 0

	○ Since arr[3][0] == 0, dp[3][0] = 0. 13. **For i = 2, j = 5**: ○ arr[2][5] = 0 ○ Since arr[2][5] == 0, dp[2][5] = 0. 14. **For i = 2, j = 4**: ○ arr[2][4] = 1 ○ dp[2][4] = min(dp[2][5], min(dp[3][4], dp[3][5])) + 1 = min(0, min(1, 0)) + 1 = 1. ○ largestSide = max(largestSide, dp[2][4]) = max(2, 1) = 2. 15. **For i = 2, j = 3**: ○ arr[2][3] = 1 ○ dp[2][3] = min(dp[2][4], min(dp[3][3], dp[3][4])) + 1 = min(1, min(2, 1)) + 1 = 2. ○ largestSide = max(largestSide, dp[2][3]) = max(2, 2) = 2. 16. **For i = 2, j = 2**: ○ arr[2][2] = 1 ○ dp[2][2] = min(dp[2][3], min(dp[3][2], dp[3][3])) + 1 = min(2, min(2, 2)) + 1 = 3. ○ largestSide = max(largestSide, dp[2][2]) = max(2, 3) = 3. 17. **For i = 2, j = 1**: ○ arr[2][1] = 1 ○ dp[2][1] = min(dp[2][2], min(dp[3][1], dp[3][2])) + 1 = min(3, min(0, 2)) + 1 = 1. ○ largestSide = max(largestSide, dp[2][1]) = max(3, 1) = 3. 18. **For i = 2, j = 0**: ○ arr[2][0] = 0 ○ Since arr[2][0] == 0, dp[2][0] = 0. **Result:** The largest square submatrix has side length 3
Output:- 3	