```
//**************************************************************
// Project #2 2D STL vector problem (10855 - Rotated Square)
// Name: Ben Diekhoff
// Data Structures and Algorithms
// Date: 02/06/2019
//***************************************************************
// This program reads in two matrices, and big one and a smaller one. Then it
// passes the smaller matrix to a function that rotates it 90 degrees and
// returns it. Each variant of the smaller matrix is compared to the big
// one to see if it is found inside. Then, the number of times each variant
//  of the small matrix is found within the big matrix is printed out.
//
//                       COMPLEXITY
// The most complex function in this matrix by far is compareMat(), which
// compares every element in the smaller matrix to a chunk of the big one.
// This function require 4 nested for-loops, so the worst case scenario for
// this program is O (N^4).
//
//*****************************************************************
/* I have written the enitre program as turned in and have not copied this
code, or parts of this code from the internet or another student.
Signature______________________________________________*/
//******************************************************************


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

// Function prototypes
vector<vector<char>> rotateMat(const short, const vector<vector<char>> &);

short compareMat(const short, const short, const vector<vector<char>> &,
    const vector<vector<char>> &);


//*******************************************************************
// main()
// Parameters: none
// Complexity: O(N^2)
// Reads in the sizes of the matrices and then reads in the actual matrices.
// Calls the rotateMat function for the smaller matrix, and then calls the
// compareMat function and prints out the results.
// This function has two nested for loops used to read in the big and small
// matrices, so its complexity is O(N^2).
//********************************************************************
int main() {
    // s1 and s2 refer to the size of the big and small matrices, respectively.
    // a1, a2, a3, and a4 refer to the number of times a rotation of the small
    //  matrix is found inside the big matrix.
    short s1, s2, a1, a2, a3, a4;
    cin >> s1 >> s2;

    // Only runs when there is a new matrix to be read in.
    while (s1 > 0 && s2 > 0) {
        vector<vector<char>> big(s1, vector<char>(s1));
        vector<vector<char>> small(s2, vector<char>(s2));

        // Read in the big matrix.
        for (short r = 0; r < s1; r++) {
            for (short c = 0; c < s1; c++) {
                cin >> big[r][c];
            }
        }

        // Read in the small matrix.
        for (short r = 0; r < s2; r++) {
            for (short c = 0; c < s2; c++) {
                cin >> small[r][c];
            }
        }

        //create new vectors for each rotation of the small matrix.
        vector<vector<char>> small_90 = rotateMat(s2, small);
        vector<vector<char>> small_180 = rotateMat(s2, small_90);
        vector<vector<char>> small_270 = rotateMat(s2, small_180);

        // Compare each rotation of the small matrix to the big matrix.
        a1 = compareMat(s1, s2, big, small);
        a2 = compareMat(s1, s2, big, small_90);
        a3 = compareMat(s1, s2, big, small_180);
        a4 = compareMat(s1, s2, big, small_270);

        // Output the number of times each rotation of the small matrix is
        // found in the large matrix.
        cout << a1 << " " << a2 << " " << a3 << " " << a4 << endl;

        // Read in the sizes of the new matrices.
        cin >> s1 >> s2;
    }

    return 0;
}


//*******************************************************************
// rotateMat()
// Parameters: 1 const short, 1 const 2D vector of chars, passed by reference
// Complexity: O(N^2)
//
// This function is passed the original small matrix by reference.
// It copies it into a new one (rotate), reading the columns and rows
// of the original into the rows and columns of rotate.
// Each time a full row is read into rotate, it's reversed, and the next row
// is read in. After the matrix is completely rotated, the function returns it.
// This function has a nested for loop, so its complexity is O (N^2).
//********************************************************************
vector<vector<char>> rotateMat(const short s2,
    const vector<vector<char>> &vect) {

    vector<vector<char>> rotate(s2, vector<char>(s2));

    for (short r = 0; r < s2; r++) {
        for (short c = 0; c < s2; c++) {
            rotate[r][c] = vect[c][r];
        }
        reverse(rotate[r].begin(), rotate[r].end());
    }

    return rotate;
}



//*******************************************************************
// CompareMat()
// Parameters: 2 const shorts, 2 const 2D vector of chars, passed by reference
// Complexity: O(N^4)
//
// This function is passed the big matrix and the small matrix, or one of its
// rotations, along with the number of rows and columns for big and small
// (s1 and s2, respectively). Small is compared to big and the total number
// of times small is found inside big is returned.
// This function uses 4 nested for loops and the entirety of each matrix is
// compared each time, so its complexity is O (N^4)
//********************************************************************
short compareMat(const short s1, const short s2,
    const vector<vector<char>> &big, const vector<vector<char>> &small) {

    short bound = 1 + (s1 - s2); // The farthest element in big that the first
                                // element of small should compare itself to

    short count = 0;  // Number of consecutive elements in small that match big
    short match = 0;  // Number of times small is found in big
    short smallElements = s2 * s2;  // Total number of elements in small

    // These two loops make sure that the small matrix is compared against
    // the entirety of the big one, without checking out of bounds.
    for (short r = 0; r < bound; r++) {
        for (short c = 0; c < bound; c++) {
            count = 0;      // reset count;

            // These loops compare the entirety of small against
            // a section of big that is the same size as small.
            for (short m = 0; m < s2; m++) {
```

```cpp
                for (short n = 0; n < s2; n++) {

                    // If an element matches, increment count
                    if (small[m][n] == big[r + m][c + n])
                    {
                        count++;
                    }
                }

                // If each element in small matches each element of big,
                // there is a match. Sometimes small can have more than
                // one match, so match is incremented each time.
                if (count == smallElements) {
                    match++;
                }
            }
        }

    }
    return match;
}
```