```
/*
    Time complexity: O(N^3)
    Space complexity: O(N^2)

    where N is the number of vertices in the input graph and
    E is the number of edges in the input graph
*/

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
public class Solution {
    static BufferedReader br = new BufferedReader(new InputStreamReader(System.in));


    public static int solve(boolean[][] graph,int n)
        {
        int cycleCount = 0;

        for (int i = 0; i < n - 2; ++i) {
            for (int j = i + 1; j < n - 1; ++j) {
                for (int k = j + 1; k < n; ++k) {
                    if (graph[i][j] && graph[j][k] && graph[k][i]) {
                        ++cycleCount;
                    }
                }
            }
        }

        return cycleCount;
        }
    public static boolean[][] takeInput() throws IOException {
                String[] strNums;
        strNums = br.readLine().split("\\s");
        int n = Integer.parseInt(strNums[0]);
        int m = Integer.parseInt(strNums[1]);

        boolean[][] graphs = new boolean[n][n];
        int firstvertex, secondvertex;

        for (int i = 0; i < m; i++) {
            String[] strNums1;
            strNums1 = br.readLine().split("\\s");
            firstvertex = Integer.parseInt(strNums1[0]);
            secondvertex = Integer.parseInt(strNums1[1]);
            graphs[firstvertex][secondvertex] = true;
            graphs[secondvertex][firstvertex] = true;
        }
        return graphs;
    }
    public static void main(String[] args) throws NumberFormatException, IOException {
        boolean [][]graphs = takeInput();

        int ans = solve(graphs, graphs.length);
        System.out.println(ans);

        }
}

/*
    An optimised solution can also be written
    whose complexity may be as follows:

    Time complexity: O(N^2)
    Space complexity: O(N + M)
    where N is the number of vertex in the graph
    and M is the number of edges in the graph
*/
```