# E. Pentagon

time limit per test: 10 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

According to the last order issued by the president of Berland every city of the country must have its own Ministry Defense building (their own Pentagon). A megapolis Berbourg was not an exception. This city has $n$ junctions, some pairs of which are connected by two-way roads. Overall there are $m$ roads in the city, no more than one between each pair of junctions.

At the moment choosing a location place for Pentagon in Berbourg is being discussed. It has been decided that Pentagon should cover the territory of five different junctions which are joined into a cycle by roads. In the order to build Pentagon a special wall will be built along the roads (with high-tension razor, high-voltage wire and other attributes). Thus, the number of possible ways of building Pentagon in the city is equal to the number of different cycles at lengths of 5, composed of junctions and roads.

Your task is to prints the number of ways of building Pentagon in Berbourg. Only well-optimized solutions will be accepted. Please, test your code on the maximal testcase.

## Input

The first line contains two integers $n$ and $m$ ($1 \le n \le 700; 0 \le m \le n \cdot (n - 1) / 2$), where $n$ represents the number of junctions and $m$ is the number of roads in the city. Then follow $m$ lines containing the road descriptions, one in each line. Every road is set by a number of integers $a_i, b_i$ ($1 \le a_i, b_i \le n; a_i \ne b_i$), where $a_i$ and $b_i$ represent the numbers of junctions, connected by the road. The junctions are numbered from 1 to $n$. It is not guaranteed that from any junction one can get to any other one moving along the roads.

## Output

Print the single number which represents the required number of ways. Please, do not use `%lld` specificator to read or write 64-bit integers in C++. It is preffered to use `cout` (also you may use `%I64d`).

### Examples

**input**

```
5 5
1 2
2 3
3 4
4 5
5 1
```

**output**

```
1
```

**input**

```
5 10
1 2
1 3
1 4
1 5
2 3
2 4
2 5
3 4
3 5
4 5
```

**output**

```
12
```