Let X be the set of all integers between 0 and n-1. Suppose we have a collection S_1, S_2, ..., S_m of subsets of X. Say an atom A is a subset of X such that for each S_i we have either A is a subset of S_i or A and S_i do not have any common elements.

Your task is to find a collection A_1, ..., A_k of atoms such that every item in X is in some A_i and no two A_i, A_j with $i \neq j$ share a common item. Surely such a collection exists as we could create a single set {x} for each x in X. A more interesting question is to minimize k, the number of atoms.

Input

The first line contains a single positive integer $t \leq 30$ indicating the number of test cases. Each test case begins with two integers n,m where n is the size of X and m is the number of sets S_i. Then m lines follow where the i'th such line begins with an integer v_i between 1 and n (inclusive) indicating the size of S_i. Following this are v_i distinct integers between 0 and n-1 that describe the contents of S_i.

You are guaranteed that $1 \leq n \leq 100$ and $1 \leq m \leq 30$. Furthermore, each number between 0 and n-1 will appear in at least one set S_i.

Output

For each test case you are to output a single integer indicating the minimum number of atoms that X can be partitioned into to satisfy the constraints.

Sample Input

```
2

5 2

3 0 1 2

3 2 3 4

4 3

2 0 1

2 1 2

2 2 3
```

Sample Output

```
3
```

```
import java.io.*;
import java.util.StringTokenizer;

public class Program {
	BufferedReader in;
	StringTokenizer str;
	PrintWriter out;

	String next() throws IOException {
		while ((str == null) || (!str.hasMoreTokens())) {
			str = new StringTokenizer(in.readLine());
		}
		;
		return str.nextToken();
	};

	int nextInt() throws IOException {
		return Integer.parseInt(next());
	};

	double nextDouble() throws IOException {
		return Double.parseDouble(next());
	};

	double nextLong() throws IOException {
		return Long.parseLong(next());
	};

	int n, m;
	int[][] a;
	int[] buv;
	int[] kilk;

	void dfs(int v) {
		buv[v] = 1;
		for (int i = 0; i < n; i++)
			if ((a[v][i] == 0) && (buv[i] == 0)) {
				dfs(i);
			}
		;
	};

	void solve() throws IOException {
		n = nextInt();
		m = nextInt();
		a = new int[n][n];
		buv = new int[n];
		kilk = new int[n];
		for (int i = 0; i < m; i++) {
			int t = nextInt();
```

```
                int now[] = new int[n];
                for (int j = 0; j < t; j++) {
                        int k = nextInt();
                        now[k] = 1;
                }
                for (int j = 0; j < n; j++)
                        for (int l = 0; l < n; l++) {
                                if ((now[j] ^ now[l]) == 1) {
                                        a[j][l] = 1;
                                        a[l][j] = 1;
                                }
                        }
        }
        ;
        int res = 0;
        for (int i = 0; i < n; i++)
                if (buv[i] == 0) {
                        res++;
                        dfs(i);
                }
        ;
        out.println(res);
    };

    void run() throws IOException {
        in = new BufferedReader(new InputStreamReader(System.in));
        out = new PrintWriter(System.out);
        int n = nextInt();
        for (int i = 0; i < n; i++) {
                solve();
        }
        ;
        out.close();
    }

    public static void main(String[] args) throws IOException {
        new Main().run();
    }
}
```