Problem A. Big Matrix

The task essentially consists of counting the number of tokens in a string and the number of lines. In C/C++, this can be done, for example, by reading the first line (for instance, using `fgets`) and counting the number of spaces in it. By increasing this count by 1, we obtain c. By reading lines until the end of the file, we find out the number of lines, thus obtaining the value r.

Problem J. Count them!

Consider the recursive solution.

We use the function `run(n, s)` which returns the count of numbers with n digits whose digit sum equals s. The function is implemented recursively:

- For $n == 1$, it returns 1 when $s < p$, and 0 otherwise (handling single-digit numbers)
- For $n > 1$, it returns value c calculated by:

```
typedef long long int LL;
LL c = 0;
for(int i = 0; i < p; i++)
    if(i <= s) c += run(n-1, s-i);
```

In the main function, since the leading digit cannot be zero, `run()` is called in this loop:

```
LL c = 0;
for(int i = 1; i < p; i++)
    if(i <= s) c += run(n-1, s-i);
```

And then we print the value of c.

But it is still not fast enough, so we need to speed it up using memoization.

We introduce a helper array `t[n][s]` to store the return values of `run(n, s)`:

- Initialize the array with -1 using `memset(t, -1, sizeof(t))`
- In the `run()` function, we cut off recursive branches with:

```
if (t[n][s] != -1) return t[n][s];
```

Problem K. Find the Property

The problem asks us to determine whether a given graph G is bipartite and, given additional edges, to find the first edge whose addition breaks the bipartiteness.

When there are no additional edges ($q = 0$), this is a classical problem that can be solved in multiple ways—using DFS, BFS, or even Union-Find (Disjoint Set Union).

The convenience of the bipartiteness problem lies in the fact that the two colors (e.g., black and white) serve the same purpose. Regardless of the initial color assigned to a vertex u, if a solution exists, the algorithm should be able to find it.

For DFS, we can start from an arbitrary vertex with an arbitrary color. During traversal, we alternate the color for each visited vertex. Issues may arise with back edges in the DFS tree. For these, we simply need to check that the color of the previously visited vertex differs from the current vertex's color.

A similar strategy applies to BFS. Each layer in the BFS tree consists of vertices of the same color. Problematic edges here are those connecting two vertices in the same layer—these must be checked for existence.

The idea of checking bipartiteness with Union-Find is slightly more complex but can directly lead to a solution for the full problem (100 points). Below, we will also explore other solutions.

We extend the graph G as follows: each vertex u will have two versions—u_{white} and u_{black}—representing the idea of coloring u white or black, respectively. Adding an edge $u - v$ means connecting $u_{\text{white}} - v_{\text{black}}$ and $u_{\text{black}} - v_{\text{white}}$. Intuitively, the existence of the edge $u_{\text{white}} - v_{\text{black}}$ is equivalent to the statement: "If u is white, then v is black", and vice versa. Problems arise when, following these statements, we derive a contradiction like "If u is white, then u is black".

The algorithm is as follows: we add all edges as described above and check that no u_{white} and u_{black} are connected for any u.

The reason we don't yet have a full solution is that, although we build the graph quickly, we check every vertex. This is resolved with the following observation: When adding an edge $u - v$ that breaks the bipartiteness of the original graph, it creates an odd-length cycle. For every vertex w in this odd cycle, w_{white} and w_{black} are connected in the same component. Leveraging this, after adding $u - v$, we only need to check the pairs $(u_{\text{white}}, u_{\text{black}})$ and $(v_{\text{white}}, v_{\text{black}})$. This yields a full solution with complexity $O((n + m + q) \log n)$ or $O((n + m + q)\alpha(n))$, depending on Union-Find optimizations.