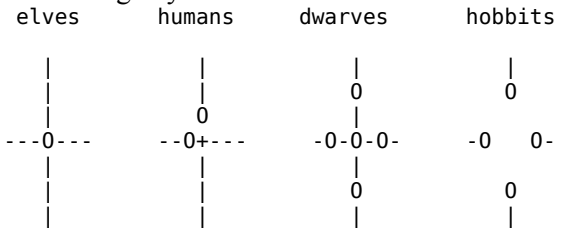
Solution sketches for Day 1

Alliances:

- Human-only inputs can be solved simply: just iterate row by row, each cell is uniquely determined when you visit it.
- Inputs with a small number of columns can be solved using dynamic programming / memoization: Process the cells row by row. At any moment the answer only depends on the following variables:
  - what is the next cell to be processed,
  - for each column, is there an alliance (edge) going down from the last processed cell?

  In this way we have O(RC*2^C) states only.
- For 100 points: it is possible to solve the task using maximum matching in a bipartite graph. Replace each village by a cluster of nodes as shown below:



  Human village is replaced by two nodes: one connected to the top and to the bottom, the other left+right. Edges which cross cell boundary represent potential alliances.

Arithmetic:

- For 2 times 2 inputs all solutions are valid, it was sufficient to replace dots by zeroes.
- The entire rectangle is uniquely determined by four independent values, such as the values in the first two rows and columns. Assign four variables to these four cells. Then each prescribed value gives you a linear equation in these four variables. Solve the system.
  (The number of variables is low, so it is possible to solve the system in integers without overflow, and only then to evaluate the variables as fractions.)
- For a solution to exist, all denominators must be reasonably small. It should be possible to solve the system using floating point numbers, and then to convert the floating point number into a matching fraction. (This may time out for batch 10.)
- It is possible to only have a system of three equations if we pick one of the input numbers as one of the variables. (This was not necessary.)

Bodyguards:

- Note that the relative order of rows and columns does not matter. So the first step is to sort the groups according to the number of bodyguards in each row/column in a descending way.
- The following simple greedy strategy works: pick the row with most bodyguards required. Place the bodyguards into columns that currently require the most bodyguards. (It can be implemented in variously efficient ways.)
- Consider the seating of the bodyguards in the auditorium in which there is the correct number of bodyguards in each column, and these occupy the first possible rows in each column. This seating is possibly invalid but it shows us a neccessary condition for a solution to exist: For any K, the total number of bodyguards currently sitting in the first K rows must not be smaller than the number of required bodyguards in the respective rows.
- It can be proved that the above set of conditions is also sufficient: if all of them are satisfied, we can start with the invalid seating above and transform it into a valid one by repeatedly moving bodyguards to lower rows.
- So a solution is to check the above conditions. We do not need to check them row by row but only at the places where the number of bodyguards changes.