Problem 270: Piece by Piece

Difficulty: Hard

Author: Brett Reynolds, Bethesda, Maryland, United States

Originally Published: Code Quest 2025

Problem Background

Our ability to produces images of our environment has progressed incredibly since the invention of the daguerreotype in 1839. Current technology allows us to record images on virtually any scale - from the atomic structure of a substance to the vast expanses of the cosmos - and at incredible resolutions. At the time this problem was written, there were dozens of photos measuring in the hundreds of gigapixels; if printed at a standard resolution of 300 dpi, some of these images would cover more than a square kilometer.

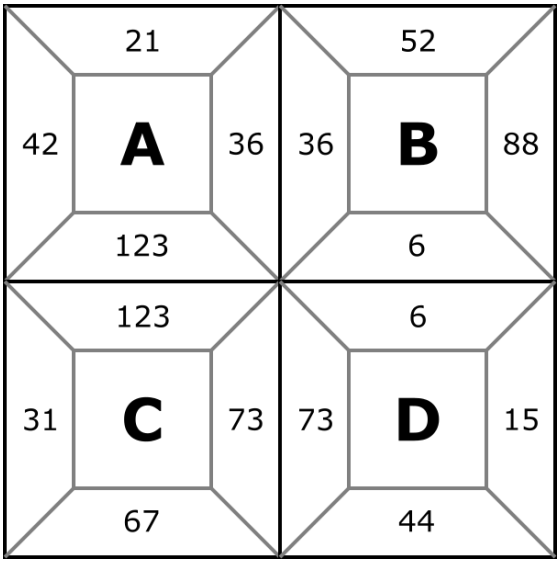
Obviously, such large images are impractical for most uses; nobody's going to print out an image that covers an area larger than a city block, and most personal computers would struggle to even store such an image, let alone do anything with it. So, as with any software problem, the solution is simple: break the problem (image) into smaller parts that are easier to work with. "Image chunking" does exactly this; it divides a larger image into many smaller, uniformly sized "chunks" that accurately represent a tiny portion of the original image. This allows computers to more easily perform tasks on the image, as they don't need to load the entire image into memory; only a small bit of it at a time. However, it's important to ensure that the computer understands each chunk's location relative to the larger image, or your work becomes meaningless gibberish.

Problem Description

Lockheed Martin's Rotary and Mission Systems is working with the National Reconnaissance Office to develop a new radar imaging system that can cover a huge area. The images generated by the radar system are so large that they can't be stored as single files; they're saved as a number of image chunks along with the information needed to reconstruct the full image. Normally, this reconstruction is fairly straightforward, as each file is given a sequential name.

Unfortunately, someone on another team was careless about clicking links in an email, and exposed your work site to a virus that scrambled all of your files around. After containing the virus and assessing the damage, you've been able to determine that all of your data is still present, but some of it has been altered. Each of the chunk files was renamed, so you no longer have an easy method to reconstruct the full image. Additionally, it appears as though some of the images were rotated, so simply putting them in the correct order again won't be enough to get a usable image. You'll have to examine each of the files to determine how they fit together so that the NRO can perform a meaningful analysis on the data.

Each chunk file contains metadata that can be used to reconstruct the full image. Fortunately, it looks as though the virus managed to use your team's rotation utility program when corrupting the files, so the metadata is still largely valid. Each side of each chunk is assigned a random number as an identifier. When two chunks are adjacent to each other, the touching edges will have the same identifier number; no other edges will have that identifier. See the image below for an example:



When a chunk appears on an edge of the image (or a corner), one (or two) of its edges will contain a unique identifier number, which does not appear in any other chunk.

As some of the chunks have been rotated, edges may not line up correctly in their current orientations; using the image above as an example, edge 36, shared between chunks A and B, may have been presented as chunk B's top edge originally.

Sample Input

The first line of your program's input, **received from the standard input channel**, will contain a positive integer representing the number of test cases. Each test case will include the following lines of text:

- A line containing the following values, separated by spaces:
  - A positive integer, H, representing the height of the image in chunks. This value will be greater than or equal to 2.
  - A positive integer, W, representing the width of the image in chunks. This value will be greater than or equal to 2.
  - A single visible ASCII character, identifying a chunk that is known to be in the correct orientation (that is, the identified chunk does not require rotation; all other chunks may or may not require rotation). For a list of possible characters, see the US ASCII Table in the reference materials; spaces will not be used.
- H * W lines, each containing data extracted from a chunk file's metadata. This includes the following values, separated by spaces:

- A single visible ASCII character that uniquely identifies the chunk. For a list of possible characters, see the US ASCII Table in the reference materials; spaces will not be used.
- An integer representing the identifier for the chunk's top edge.
- An integer representing the identifier for the chunk's right edge.
- An integer representing the identifier for the chunk's bottom edge.
- An integer representing the identifier for the chunk's left edge.

Within each test case, the resolution of the image (H * W) will not exceed 90 chunks.

```
2
2 2 A
A 21 36 123 42
B 36 52 88 6
C 73 67 31 123
D 6 15 44 73
3 3 %
$ 5 32 58 46
& 70 33 62 46
@ 71 44 80 1
% 80 19 83 58
^ 88 19 95 84
( 54 67 88 12
! 9 1 32 4
# 7 95 44 2
* 83 67 89 70
```

Sample Output

For each test case, your program must print H lines, each containing W ASCII characters representing the correct ordering of chunks within the full image. Lines should be printed in top-to-bottom order, with each line listing chunks within that row in left-to-right order.

```
AB
CD
!@#
$%^
&*(
```