Rune Activation

Objective
Give students practice with backtracking/recursion.

Story
At the oasis you found a secret entrance to a lost tomb of a ruler of an ancient civilization. On the walls are written an ancient spell that can activate and deactivate old enchantments throughout the structure. Upon reaching an empty room you notice a rectangular grid of square tiles each containing a glyph on the floor. You cast the spell you learnt a few times and notice an interesting behavior.

All glyphs in the room are either in an activated or deactivated state. While standing on one of the floor glyphs you can cast the spell. Casting the spell while standing on the glyph will invert the activation/deactivation state of the glyph you are standing on and the adjacent glyph(s). A glyph is considered adjacent to another, if the glyphs share a side.

An engraving on a pedestal at the front of the room shows what you assume to be a required activation/deactivation pattern. You want to know the minimal amount of spell casts required to convert the floor into the picture engraved on the pedestal.

Problem
Given a current activation/deactivation glyph pattern and a desired activation/deactivation pattern determine the fewest number of spell casts to change the current pattern into the desired one, where each spell flips the state of the current glyph and adjacent glyph(s). Note that it might not be possible to cast spells to correctly convert the original glyph pattern into the desired one.

Input
Input will begin with a line containing 2 integers, ***r*** and ***c*** ($1 \le r, c \le 15$), representing the number of rows and columns of the rectangular grid of glyphs, respectively. The following ***r*** lines will each contain a ***c*** character length string. The ***i***-th character of the ***j***-th string (of the first ***r*** strings) will either be an uppercase ‘`A`’ or an uppercase ‘`D`’ representing the ***j***-th glyph of the ***i***-th row as **currently** activated (‘`A`’) or deactivated (‘`D`’) glyph. The following ***r*** lines will each contain a ***c*** character length string. The ***i***-th character of the ***j***-th string (of the first ***r*** strings) will either be an uppercase ‘`A`’ or an uppercase ‘`D`’ representing the ***j***-th glyph of the ***i***-th row as a glyph **desired** to be activated (‘`A`’) or deactivated (‘`D`’).

Output
IF the conversion is possible, output should contain ***r*** lines. Each line will contain a string of ***c*** characters either of an uppercase ‘`C`’ or a hyphen (‘`-`’). A ‘`C`’ in the ***j***-th spot of the ***i***-th row represents that a spell should be cast on the ***j***-th glyph of the ***i***-th row. A hyphen (‘-’) represents

that a spell should not be cast in that spot. The output should be given such that the number of spell casts is minimized. If multiple minimal casts are possible, output any valid minimal casts.

IF the conversion is not possible, output a -1 instead.

Sample Input	**Sample Output**

`3 4`
`ADDA`
`DADA`
`ADAD`
`AAAA`
`DDDD`
`AAAA`
	`CCC-`
	`--CC`
	`CCCC`

`2 2`
`AD`
`DA`
`AA`
`DA`
	`CC`
	`-C`

Explanation

Case 1

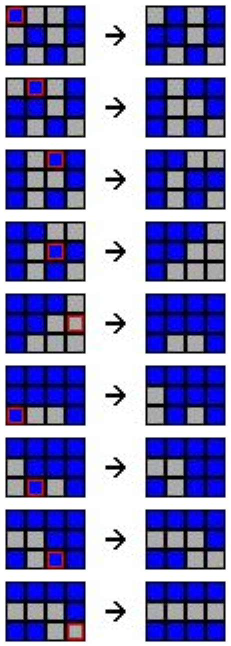
The first line of input indicates that there will be 3 rows of 4 columns of glyphs. The first 3 lines indicate the starting grid. The following grid represents the starting grid,

Active	**Deactive**	**Deactive**	**Active**
Deactive	**Active**	**Deactive**	**Active**
Active	**Deactive**	**Active**	**Deactive**

The next 3 lines initiate the desired grid, seen by the following grid,

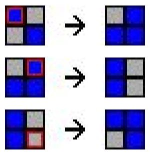
Active	**Active**	**Active**	**Active**
Deactive	**Deactive**	**Deactive**	**Deactive**
Active	**Active**	**Active**	**Active**

If we draw the active glyphs as blue and the deactivated glyphs as gray we can see the following behavior by casting the spell in the locations denoted in the sample output.



Case 2

Here’s how case 2 would work by casting spells at the indicated locations.



Hints

Casting Order: The order in which the spells are cast does not matter.

Backtracking Structure: Use the recursive sudoku solver style of traversing the grid. There are only 2 options at each spot (sudoku had 9 options): cast or don't cast.

Noticing Mistakes: If you ever find that a glyph does not match the target grid's glyph, but you cannot change the state of the glyph, then the glyph grid has reached an invalid state! By traversing the board in the sudoku solver pattern the spots that cannot be changed are the following,

CANNOT CHANGE	CANNOT	CANNOT	CANNOT	CANNOT	CANNOT
CANNOT	CANNOT	CANNOT	CAN CHANGE	CAN	CAN
CAN	CAN	CAN	HERE (TO DO)	CAN (TO DO)	CAN (TO DO)
CAN (TO DO)	CAN (TO DO)	CAN (TO DO)	CAN (TO DO)	CAN (TO DO)	CAN (TO DO)
CAN (TO DO)	CAN (TO DO)	CAN (TO DO)	CAN (TO DO)	CAN (TO DO)	CAN (TO DO)

Once you make a decision on a cell consider "Which cell(s) will no longer be capable of changing?"

Extra Challenge (not needed for full points): Checking if the grid is done, will probably take O(Rows X Columns), see if you can get the check to be O(1)

Grading Criteria

- Good comments, whitespace, and variable names
  - 15 points
- No extra input output (e.g. input prompts, “Please enter the number of friends:”)
  - 10 points
- Implement a way to check if a grid matches the desired grid
  - 10 points
- Make something to check if no solution was found
  - 5 points
- Write the bruteforcer to try all (reasonable) outcomes
  - 10 points
- Programs will be tested on 10 cases
  - 5 points each

No points will be awarded to programs that do not compile using javac.

Sometime a requested technique will be given, and solutions without the requested technique will have their maximum points total reduced. For this problem use the backtracking technique to solve the problem. ***Without this programs will earn at most 50 points!***

Any case that causes a program to return a non-zero return code will be treated as wrong. Additionally, any case that takes longer than the maximum allowed time (the max of {5 times my solutions time, 10 seconds}) will also be treated as wrong.

No partial credit will be awarded for an incorrect case.