# ACM Asia Regional (Kanpur Site) Programming Contest
# December 13, 2012

## Instructions

There are **Eleven (11)** problems for each team to be completed in **Five hours**. Standard Input and Output files are to be used for each problem. If you test your program using PC$^2$, it will automatically redirect input from the sample input file to your program. Output must correspond exactly to the provided sample output format, including (mis)spelling and spacing. Multiple spaces will not be used in any of the judges' output, except where explicitly stated. A copy of the problem set will be available at **/users/acm/sw/problems.pdf**, during the contest. Sample input and corresponding output files for respective problems can be seen in the directories **/users/acm/sw/inputs** and **/users/acm/sw/outputs** respectively, during the contest.

Your solution to any problem should be submitted for judging using the PC$^2$ software only. Once you have submitted the solution, it will reach the judges. The time it takes for your problem to be judged will depend, among other things, on how busy the judges are. Once your submission has been judged, you will receive a message through PC$^2$ indicating the judgment. The judgment may be "Yes", meaning that your submission was judged to be correct. Otherwise you will get a message indicating the problem with your program. For example, the message may be "Incorrect Output", "Output Format Error", "Compilation Error", "Runtime Error", "Run Time Limit Exceeded" etc.

When submitting a program via PC$^2$, you are required to specify a primary source file and other source files (please see PC$^2$ documentation for details). If you are writing your programs in C or C++, please make sure that you have one primary source file from which all other source files are INCLUDED. Do not have several source files that need to be linked together. This main file in which all other files are included would be the primary source file for the program. If you are writing your programs in Java, please make sure that the name of the file containing the main class is the same as the name of the main class with a **.JAVA** suffix. This is your primary source file for the program.

You can use any of the standard library functions that your chosen program programming language provides. In addition, you can use the math library in C/C++. You cannot use any other library that requires an extra flag to be passed to the compiler command. If you do this, the judges will probably find a code "compilation error" in your program. Your program is not permitted to invoke any external programs. For example, you cannot use in C the call system ("grep xyz abc") to determine whether the file abc contains an occurrence of the string xyz. *Violation of this rule may lead to disqualification from the contest.*

Programming style is not considered in this contest. You are free to code in whatever style you prefer. Documentation is not required. The judges will only test whether the input-output behavior of your program is correct or not. *If your program takes more than 2 minutes to execute for some input, it will be assumed to have gone into an infinite loop and judged incorrect.* A problem is considered as correctly solved when it is accepted by the judges. The judges are solely responsible for accepting or rejecting submitted runs. The regional contest director and judges are empowered to adjust for or adjudicate unforeseen events and conditions. Their decisions are final.

Teams are ranked according to the most problems solved. Teams who solve the same number of problems are ranked by least total time. The total time is the sum of the time consumed for each problem solved. The time consumed for a solved problem is the time elapsed from the beginning of the contest to the submittal of the accepted run plus 20 penalty minutes for every rejected run for that problem regardless of submittal time. There is no time consumed for a problem that is not solved.

No team is allowed to bring any printed materials in the contest area. Further, the team is not allowed to discuss/ talk with any other team by any means whatsoever, during the contest period. *Any such attempt, if it is detected, may lead to immediate disqualification of all the teams involved.*

# PROBLEM A
# DATA COMPRESSION

Input file: stdin
Output file: stdout

A picture may be worth a thousand words, but on the internet, sending a thousand words cost you less than sending a picture. It is better still, to send a compressed text. So, Thrifty Mailer decides to save even more with data compression, using the *Move-To-Front Transform*, described as follows.

Let the text consist of 26 letters of the alphabet and the "space" symbol (denoted here by '_'). The alphabet is first listed in the canonical order `abcdefghijklmnopqrstuvwxyz_`. Let us denote this with O. Suppose we want to send a text w[0]w[1]…w[n-1]. Instead of sending the text, we send a string of numbers described as follows.

The first number is the index of the letter w[0] in the initial alphabet array. We then change O by bringing that letter forward. For example, if the first letter is 'c', the index is 2, and the new alphabet array O is `cabdefghijklmnopqrstuvwxyz_`.

We then repeat the process for the substring starting with w[1], until there are no more strings to send. Thus the compressed version of "cat" is "2,1,19". Compression occurs when the more frequently used letters are towards the beginning, leading to a string of very small numbers.

Thrifty Mailer implements this algorithm – being what he is, he decides to save on the comma separators. So he would want to send "2119" for cat. This is obviously ambiguous, since it can be parsed into "2,11,9" and "21,1,9" as well. So in addition, Thrifty Mailer will send 2 additional pieces of information – how many letters there are, and what the final letter of the compressed string is.

So Thrifty Mailer sends cat as follows "2119 3 t".

Unfortunately, the scheme is still ambiguous. Could you help him to write a program which decompresses the string and gives back the input if there is one such unique string, and tells "AMBIGUOUS" when there are multiple such strings, and "ERROR" when the given encoding is invalid and cannot be the compression of any piece of text?

<u>Input</u>

There are several lines of input. The first line is a number "n" specifying how many compressed inputs are given. This is followed by n lines, each as follows. There is a string of numbers specifying the compressed indices, followed by a space, followed by a number specifying the number of characters in the input, followed by a space, finally with a single character specifying the last letter in the input. The initial string will contain at most a million characters.

Output

The output consists of several lines, one for each input. If the input is valid, the line should be the decompressed input (note that this is in small letters). If the input is ambiguous, then the program should print AMBIGUOUS on that line (in capitals). If the input line contains invalid compressed string, then the program should print ERROR (in capitals).

Sample Input

```
4
0 1 b
0 1 a
121 2 a
1111 3 l
```

Sample Output

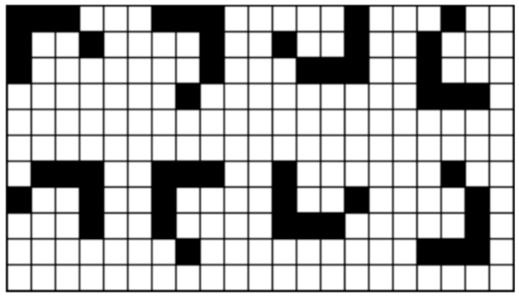
```
ERROR
a
ma
AMBIGUOUS
```

# PROBLEM B
# CONSTELLATIONS

Input file: stdin
Output file: stdout

Stars appear in clusters of various shapes. A cluster is a non-empty group of neighboring stars, adjacent in horizontal, vertical or diagonal direction. A cluster cannot be a part of a larger cluster.

Clusters may be similar. Two clusters are similar if they have the same shape and number of stars, irrespective of their orientation. In general, the number of possible orientations for a cluster is eight, as Figure below exemplifies.



The night sky is represented by a sky map, which is a two-dimensional matrix of 0's and 1's. A cell contains the digit 1 if it has a star, and the digit 0 otherwise.

Given a sky map, mark all the clusters with lower case letters of the English alphabet. Similar clusters must be marked with the same letter; non-similar clusters must be marked with different letters.

You mark a cluster with a lower case letter by replacing every 1 in the cluster by that lower case letter.

## Input

The first two lines of the input contain, respectively, the width W and the height H of a sky map. Assume that the sky is not more than 100 cells wide or high. The number of clusters is at most 500 and the number of dissimilar clusters is at most 26.

The sky map is given in the following H lines, of W characters each.

## Output

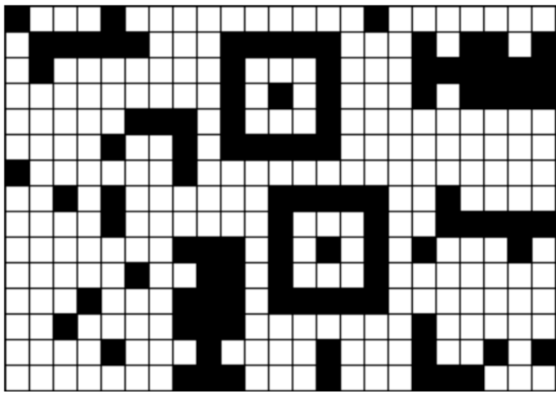
Print the modified skymap with 1s replaced by the letters representing the cluster to which a star belongs. It is depicted in the following figure for illustration purpose only. Similar clusters must be labeled with the same letter. Dissimilar clusters should be labeled with different letters.

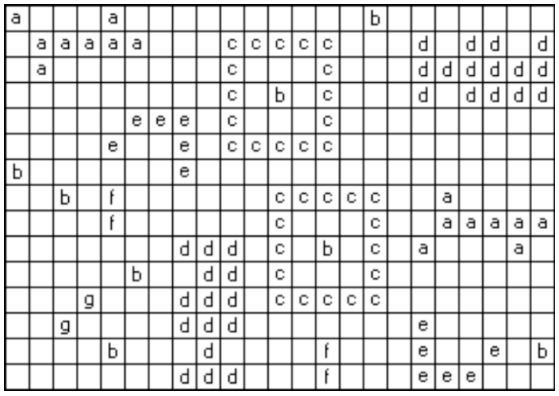
Sample Input

```
23
15
10001000000000010000000
01111100011111000101101
01000000010001000111111
00000000010101000101111
00000111010001000000000
00001001011111000000000
10000001000000000000000
00101000000111110010000
00001000000100010011111
00000001110101010100010
00000100110100010000000
00010001110111110000000
00100000111000000001000000
00001000100001000100101
00000000111000100011100
```

Sample Output

```
a000a0000000000b0000000
0aaaaa000ccccc000d0dd0d
0a0000000c000c000dddddd
000000000c0b0c000d0dddd
00000eee0c000c000000000
0000e00e0ccccc000000000
b000000e000000000000000
00b0f000000ccccc00a0000
0000f000000c000c00aaaaa
0000000ddd0c0b0c0a000a0
00000b00dd0c000c0000000
000g000ddd0ccccc0000000
00g0000ddd0000000e00000
0000b000d0000f000e00e0b
0000000ddd000f000eee000
```


Input visualization


Output visualization

# PROBLEM C
# CROSSTALK

Input file: stdin
Output file: stdout

Morse code is a way of communicating text encoding symbols using dots (**.**) and dashes (**-**). The International Morse Code (IMC) design was done keeping in mind certain statistical properties of English – for example, since E is the most frequently occurring English letter, it is assigned a short code (**.**). The IMC, in turn, leads to some famous phrases – an example is **...---...**, which is S(**...**)O(**---**)S(**...**).

For this question, we fix the language of dots and dashes, not necessarily the IMC. Often, in emergency communications, the same message will be repeated to ensure with high degree of confidence, that the message gets through to the receiver. Let x be a nonempty string of symbols. Let $x^k$ to denote k copies of x concatenated together. We say that a string x' is a *repetition* of x if it is a **prefix** of $x^k$ for some number k. ex: x'=**-.--.--.--.--.--.** is a repetition of x=-.-.

In long-distance communication, there is a small chance of “cross-talk” – when two messages get mixed with each other. We say that a string s is an *interleaving* of x and y if its symbols can be partitioned into two (not necessarily contiguous) nonempty subsequences s' and s'', such that s' is a repetition of x and s'' is a repetition y. (Each symbol in s must belong to exactly one of s' or s'').

You have to tell YES if s is an *interleaving* of x and y else NO.

Input

First line contains number of test cases T. For each test cases , there are 3 lines s,x,y. The strings contain at most ten thousand characters.

Output

For each test cases output should be YES if s is an interleaving of x and y else NO in a newline.

Sample Input

```
1
-...-.-.-
-.-
..
```

Sample Output

```
YES
```

# PROBLEM D
# A HEXAGONAL COMPLEX

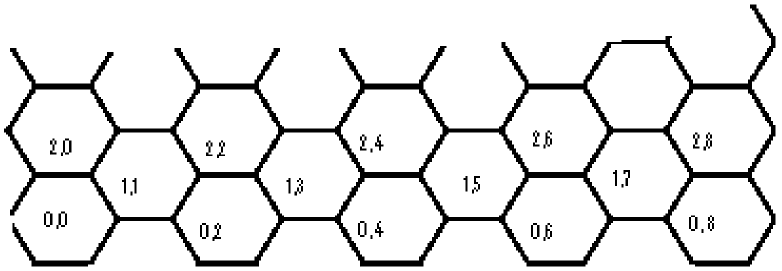
Input file: stdin
Output file: stdout

A recent storm has felled several cell phone towers in Hexagonville.

Aid and relief workers have set up a rudimentary network to communicate with each other and the outside world. Their simple idea is to station a volunteer near that location of the city with maximum number of towers still remaining. The remaining volunteers would communicate with X via handsets. X would then use the mobile network available from his site.

The *distance* between any two hexagons I and J in the network is the minimum number of hexagons to be traversed from I to reach J. (In particular, the distance from any cell to itself is 0.) Since the towers are not entirely reliable, the engineer has decided to position himself at that hexagon which is closest to the maximum number of active cells. In case there are multiple possible cells, the engineer always positions himself at the south-westernmost such cell.

For this question, you are given a list of active hexagons in a hexagonal grid. The grid consists of a number of hexagons numbered as follows. The southernmost row of hexagonal grids has row 0, and row numbers increase northwards. The westernmost column of hexagons has column number 0, and column numbers increase eastwards. (See the following figure). Note that even-numbered rows contain only even numbered columns and odd-numbered rows contain only odd-numbered columns.



<u>Input</u>

The input consists of several lines. The first line contains a number N indicating the number of grids to solve for. This is followed by N sets of inputs.

The first line of each set specifies the <u>distance</u> that the volunteer can walk. The distance will be at most 2147483647. The second line of each set says how many active hexagons there are in the network, say $M_i$. This is followed by $M_i$ lines, each containing two numbers (each less than 2147483647) separated by a space indicating the row and the column number of the active hexagonal grid. There are at most ten thousand active hexagons in any grid.

Output

The output consists of N lines. Corresponding to each set of inputs, there is a line which specifies two numbers separated by a space – this indicates the hexagon that the volunteer should be at to be closest to the maximum number of active towers, within his walking distance.

Sample Input

```
2
1
3
0 0
2 0
2 2
3
8
2 0
2 2
2 4
6 2
8 0
10 6
11 1
13 3
```

Sample Output

```
1 1
10 4
```

# PROBLEM E
# THE PAIN OF DIMENSIONALITY

Input file: stdin
Output file: stdout

You are given a chain consisting of N (N can be represented with an *int*) links, joined end-to-end in a straight line. The lengths of the links are 1,2,…, N in sequence - The link with length 1 is connected to the link with length 2, the chain with length 2 is connected to the links with length 1 and 3, and so on.

The connection between the links allows movement in any direction. The goal of this problem is to see whether various polytopes can be constructed using the given chain.

For this problem, you are interested in polytopes with the following constraint. You are given a number $K, K \leq N$. The object you have to construct must be such that any K adjacent edges must be orthogonal to each other (Note that this is possible only if you have at least K dimensions).

Your task is very simple. Given such an N and K, you have to determine whether a chain with link lengths 1,2, …, N, can be used to construct a polyhedron with any K adjacent sides being mutually orthogonal.

<u>Input</u>

The input consists of several lines. The first line M specifies the number of inputs there are. This is followed by M lines, each formatted as follows.

Each line has two positive numbers N and K, separated by spaces. N is always at least K. (N can be represented with an *int.*)

<u>Output</u>

The output consists of M lines, one for each input. For each input, the corresponding output line should print `YES` if it is possible to construct a polyhedron satisfying the condition, and `NO` otherwise.

<u>Sample Input</u>

```
2
3 2
8 2
```

<u>Sample Output</u>

```
NO
YES
```

# PROBLEM F
# SUCCINCT RANDOMNESS

Input file: stdin
Output file: stdout

At a small party, Professor Dicius is in charge of randomly allotting guests to one of $2^N$ (N is at most 20) tables. The professor's favourite randomness-generating-device is the die. But he absent-mindedly happened to wear the wrong coat today and all his dice are at home. None of the guests carry dice about them. Dicius was almost resigned to the idea of throwing a coin for N times for each guest, and assigning the table according to the outcomes. This of course would be an intolerably long procedure and would spoil the party.

Dicius instead decides on the following "pseudo-random" scheme. He would write down a bit string of length $2^N$ which contains all N-long string exactly once, if you look for strings in a sliding fashion, with wraparound. For example, the string `0110` contains the strings `01`, `11`, `10`, and `00`, from left-to-right, the last pattern being produced by looking for a 2-long string by wrapping around the pattern. Such a string is called a *succinct random string*. Thus, the first guest would be seated to table 1, the second to table 3, the third to table 2, and the fourth to table 0.

After the first $2^N$ guests are seated according to this scheme, Dicius would repeat the procedure, hoping that the guests are too distracted to notice the periodicity. Dicius assures you that the procedure is random enough, and there are such strings for every such length $2^N$.

Your task as Dicius' helper is very simple. Given a $2^N$-long binary string, you have to determine whether every table will have guests assigned to it. If the given string does not have this property, you have to indicate that the string is invalid.

## Input

The input consists of several lines. The first line is a number M which specifies how many inputs are present. This is followed by M lines, each consisting of a single binary string, at most $2^{20}$ characters long.

## Output

The output consists of M lines, one for each input string. If the corresponding input string is a *succinct random string*, then you should output `VALID`. Otherwise, you should output `INVALID`. (both in uppercase)

## Sample Input

```
3
0110
0011
01100
```

## Sample Output

```
VALID
VALID
INVALID
```

# PROBLEM G
# IT'S A JUNGLE OUT THERE

Input file: stdin
Output file: stdout

The big forest of Aranya occupies a vast expanse, filled with wild animals. The forest is represented by an NxM matrix. Each cell in the forest is marked either empty, or occupied by a lion, or occupied by a tiger. You are in charge of estimating the tiger population in the jungle.

Tigers, as you may be aware of, jealously guard their territory. If a tiger occupies A[i][j], then no tiger may occupy any of the cells A[i-2][j-2], A[i-2][j+2], A[i+2][j-2] or A[i+2][j+2], with one exception to this rule – if there is a lion in the cell A[i-1][j-1], A[i-1][j+1], A[i+1][j-1] or A[i+1][j+1], then a tiger can occupy A[i-2][j-2], A[i-2][j+2], A[i+2][j-2], or A[i+2][j+2], **respectively**.

The exact positions of lions in the jungle are known. By sampling, you know some of the positions of the tigers in the jungle. You have to estimate the maximum number of tigers in all the **unsurveyed parts** of the jungle, given this information. You can assume that the given configuration conforms to the constraints of tiger habitat.

## Input

The input consists of several lines. First line contains number of test cases N. For each test cases there are two integers N and M, specifying the size of the array NxM. N and M are at most thousand. Next, there are N lines with M characters each (with no separation). A character T indicates the presence of a tiger, character L indicates the presence of a lion and E indicates that the cell is empty.

## Output

For each test cases there is a line specifying the maximum number of tigers that can live in the empty cells. (Note that the tigers which are known should not be counted.)

## Sample Input

```
2
3 3
TET
TLE
EEE
4 3
ETT
TLE
EEE
EEE
```

## Sample Output

```
5
6
```

# PROBLEM H
# A JOYOUS BANQUET

Input file: stdin
Output file: stdout

Wedding bells are ringing in the house. The date, venue and the time of the wedding, and the subsequent feast, have all been decided. Now it is time for the most time-consuming tasks of all, that of inviting the friends and relatives for the wedding and the feast.

Now, here's the problem – you know that certain families that you plan to invite, do not get along with each other. A family has at most 4 members, and at least one member. Two families I and J are incompatible, if at least one member of I and one member of J are uncomfortable if their respective families sit next to each other, either in the wedding or at the feast. The wedding hall, fortunately has enough seats that this does not pose a problem.

The banquet hall has a limited seating capacity, ad you may have to serve several rounds to serve all the guests. You do not want incompatible families to be seated next to each other on the banquet table (the table is a long table, guests sit only one one side, and the table is not circular). To reduce expenditure in hiring the hall, you want to serve as few rounds as you can, while maintaining that only compatible families are seated next to each other. A family appears only as a whole. Also, they may be seated in multiple rounds. It goes without saying, of course, that every family has to be served.

## Input

The input consists of several lines. The first line specifies the number N of sets of inputs. This is followed by N sets of inputs, each of the following format:

The first line in each **set** specifies the number M (≤ 50) of the families. This is followed by M lines of inputs describing the M families. Each line consists of several numbers separated by **spaces**. The first number specifies the number of members F in the family. This is followed by F numbers, specifying the family members. This is followed by a number C describing the number of persons that some member in the family dislikes. Subsequently, there are C numbers specifying the guests disliked by some family member. No family is incompatible with itself. Also, if a family I is incompatible with J, then J is incompatible with I as well.

Output

The output consists of N lines, one line for each set of inputs. For each **set** of inputs, you should output the minimum number of rounds of banquet rounds required.

Sample Input

```
2
5
1 0 0
1 1 0
1 2 0
1 3 0
1 4 0
6
2 1 2 4 20 30 40 50
2 10 11 2 21 52
2 20 21 4 1 11 40 51
2 30 31 3 1 41 51
2 40 41 3 1 20 30
3 50 51 52 4 1 10 20 30
```

Sample Output

```
1
2
```

# PROBLEM I
# ROBBER'S COVE

Input file: stdin
Output file: stdout

The seacoast of Spelunksgard is flanked by a rugged mountain range with an extensive network of caves. For centuries, robbers have hidden treasure and themselves in these caves to elude the hands of the law. The caves have been extensively surveyed in modern times, and their structure has been well-understood.

There are two kinds of caves in the cave complex. A *cavern* is one where a person can hide loot, or stay. A *tunnel* is a narrow passageway between two caverns. The cave complex consists of tunnels and caverns. Due to the nature of the rocks in the hill, no cavern is situated directly above another cavern, although many tunnels are stacked on top of each other. Air shafts have been constructed in modern times, from each cavern, all the way up to the face of the hill. The cave also has a surveillance mechanism that has recently been set up, whereby every cavern and tunnel can be watched from the outside by the police.

On this occasion, policemen have determined that an armed robber (without loot), is hiding in the complex. A policeman takes a finite (nonzero) amount of time to move from one cavern to another. Since nothing much is known about the robber, the police are assuming that the robber can move from any cavern to another along a path (consisting of caverns and tunnels connecting caverns), to any other cavern with infinite speed. However, the robber cannot move through a cavern with a police officer in it.

The police officers can track the robber's movements. They want to capture the robber – that is, occupy the same cavern as the robber. If a policeman directly enters the cavern with the robber in it, the robber may hear the policeman entering through the airshaft and may escape to another cavern if there is an escape route available. You have to determine the minimum number of policemen that will ensure capture of the robber.

Input

The input consists of several lines. The first line specifies N, the number of distinct cave complexes to solve for. This is followed by N sets of inputs of the following form:

The first line contains a number M (≤ 500) specifying the number of tunnels in the cave complex. This is followed by M lines each containing two numbers I and J (each ≤ 30), separated by a space. This denotes two caverns I and J connected by a tunnel.

## Output

The output should contain N lines, one for each cave complex. The output for a given cave complex is the minimum number of police officers needed to trap a robber.

Sample Input

```
1
6
0 1
1 2
2 3
0 3
1 3
0 2
```

Sample Output

```
4
```

## PROBLEM J.
## A CHANGE IN CHANGE

Input file: stdin
Output file: stdout

Coin mintage is a venerable field with many nice puzzles. Every government wants to mint a very small number of denominations, with the constraint that every integer must be expressible using one or more coins of available denominations. The problem is trivial if 1 is one of the available denominations.

The republic of Bubblistan has got itself into a bind with its innovative financial policy. Large inflation has led to a situation where the metal to mint 1 saipa (the currency of Bubblistan) costs more than the value it represents. The government has to withdraw from minting any more 1 saipa coins.

Bubblistan has decided on a small number of new denominations to mint. However the government is now worried that there may be some values which cannot be represented by the new set of denominations.

In this question, you are given a set of denominations that have been chosen fort minting. You have to compute what the largest amount of currency that cannot be expressed using this set of denominations, is. If no upper bound exists on inexpressible amounts, you have to output INFINITY.

Input

The input consists of several lines. The first line specifies N, the number of sets of denominations. This is followed by N lines of input, of the following form:

An input line consists of a sequence of numbers separated by a **single space**. The first number M (≤ 10) specifies how many denominations are there in the currency system. This is followed by M numbers which represent the available denominations. (Each denomination can be represented with an *int*.)

Output

The output consists of several lines. For each of the N input sets, you have to output the maximum value that cannot be expressed using the coins of the denominations in the input. If there is no upper bound on the set of expressible values, then you should print `INFINITY`.

Sample Input

```
3
1 2
2 2 3
3 6 10 15
```

Sample Output

```
INFINITY
1
29
```

# PROBLEM K
# CROSSWORD

Input file: stdin
Output file: stdout

Acme Courant Management, who runs the town paper, is in dire straits. Circulation is down, and it is wondering how to draw a reluctant audience into subscribing to the paper. Being a university town, there are a lot of people who like puzzles, and good puzzles attract good readership. Of course, there is a Sudoku generating algorithm that the paper has added which does a very good job of classifying puzzles generated into easy, medium and hard. Could, however, more puzzles be automatically generated?

An editor gets the bright idea that the newspaper could generate crosswords by a program. This is a considerably different cup of tea, however, since as any crossword aficionado knows, good crosswords have very good "cryptic" clues. The editor, however, hits upon a fair enough first pass. Here's his scheme of automated crosswords.

You are given a dictionary of synonyms – each entry in the dictionary is a list of words, all of which have the same meaning. There are several such entries in the dictionary. Unlike actual English, each word has a unique meaning, so it appears in at most one of these lists.

Each crossword **clue** consists of an anagram (rearrangement of the letters) of some word in the dictionary. The **solution** to the clue is a synonym of the anagram. There may be multiple solutions to a given clue, but the entire crossword is guaranteed to have a unique solution.

Clues are of two kinds – **across** clues, and **down** clues. The solutions to the across clues are filled in from left-to-right on the crossword grid, and the solutions to the down clues are filled in from top-to-right on a crossword grid.

The clues are numbered from 1 to 10 across and 1 to 10 down. (Not all numbers may be present, since the number of clues depends on the crossword grid.)

You have to outdo the editor by writing a program that can automatically solve any crossword generated by the editor's ingenious mechanism. The constraint to be satisfied is that when an across clue and a down clue share solution cells, then the letters in the across solution and the down solution should agree on those cells. The crossword is guaranteed to have a unique solution.

<u>Input</u>

The input consists of several lines.

The first line is a number, say N describing how many lists (of synonyms) are present in the dictionary.

(The dictionary) This is followed by N lines, each consisting of several strings separated by a whitespace. A string is a contiguous, non-empty sequence of lowercase English letters.

(The crossword clues) This is immediately followed by a line with 2 numbers I & J, separated by a space, denoting the number of Across clues and Down clues. I and J are at most 100.

The next I lines provide the across clues. The subsequent J lines provide the down clues. Each clue is formatted as follows.

Each clue is a number, followed by a space, followed by a clue word.

(The crossword grid) This is followed by an integer B specifying the size of the crossword grid. This is followed by a BxB matrix of integers. B is at most 100.

A -1 in any cell in the matrix indicates a dark cell in the crossword where no letter of the solution can appear. (See the following figure for illustration)

A contiguous subrow of non-negative numbers starting with a positive number *r* indicates the space where the solution to the *r*th across clue is to be filled in. In a row, a 0 indicates that the cell is not the start of either an across or a down clue. A positive number in any cell in the subrow except the starting cell indicates the beginning of a down clue from that cell.

A contiguous subcolumn of non-negative numbers with number *c* indicates the space where the solution to the *c*th down clue is to be filled in.

<u>Output</u>

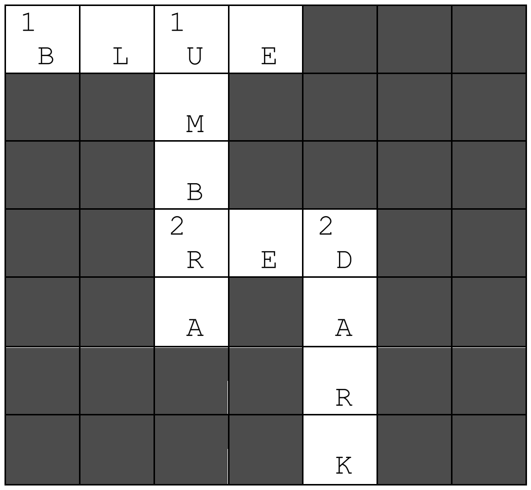
The output should consist of I+J lines. The first I lines are the solutions to the across clues, in the order of specification of the across clues. The next J lines are the solutions to the down clues, again in the order of specification of the down clues.

<u>Sample Input</u>

```
4
BLUE AZURE
UMBREA SHADOW SHADE ECLIPSE
RED MAROON
DARK BLACK NOIR
2 2
1 ZAURE
2 DER
1 CIPEELS
2 BLACK
8
 1 0 1 0 -1 -1 -1 -1
-1 -1 0 -1 -1 -1 -1 -1
-1 -1 0 -1 -1 -1 -1 -1
-1 -1 0 -1 -1 -1 -1 -1
-1 -1 2 0 2 -1 -1 -1
-1 -1 0 -1 0 -1 -1 -1
-1 -1 -1 -1 0 -1 -1 -1
-1 -1 -1 -1 0 -1 -1 -1
```

<u>Sample Output</u>

```
BLUE
RED
UMBRA
DARK
```



Crossword Visualization