```
4 5
5 6
36123
1
2374859 3029382
17
1
3 18132
```

Output for the Sample Input

```
2
13195
13
```



Nejvyssi zisky (The Highest Profits)

(**file name:** `nejvyssi.c, nejvyssi.p, nejvyssi.C`)

This program will not be part of the KOKODáKH collection, but it may be more important than the games themselves. It will serve to the marketing staff to find suitable procedures for promoting and selling the collection. So try to do your best.

Extensive marketing case study tried to prove that the total income from selling a product is polynomial function of number of satisfied customers. Experiments showed that the real results are not exactly the same which you can obtain using the function. However, this method is widely used. The reason is probably nonexistence of better solution. Let's denote the number of satisfied customers y, than we can express profits (denoted x) as

$$x = P(y) = a_0 + a_1.y + a_2.y^2 + ... + a_m.y^m$$

The number of satisfied customers depends on the price of a product. Again, there is hypothese that this dependence is polynomial. If we denote the price z, we can write

$$y = Q(x) = b_0 + b_1.z + b_2.z^2 + ... + b_n.z^n$$

Coeficients a_i and b_i strongly depend on the season of the year, the moon phase, the purchasing power of customers, inflation rate and hunderds of other parameters. Besides on the kind of product and its quality, of course. In the past there was lot of effort put into the reserarch of these parameters. For various combinations of input parameters, the coeficients are stated in Pyshwejc's marketing tables. It is not thus difficult to find out their values. But the degree of polynoms is usualy very high. It is very difficult to substitute one polynom into the other and to compute the dependency of the profit on the price. This dependency is usually crucial for us to set the right price.

Your goal is to write the program which can substitute the polynom Q into the polynom P and determine the restulting polynom R indicating dependency of the profit on the price:

$$x = R(z) = c_0 + c_1.z + c_2.z^2 + ... + c_p.z^p$$

Input Specification

The input consists of Z assignments. The number of them is given by the single positive integer Z appearing on the first line of input. Each assingement constist of three lines. On the first line there are two integers m and n ($0 <= n,m <= 100$) separated by space. These numbers give the degree of polynoms Q and P. On the second line there are $m+1$ integers a_0 ... a_m. These numbers are coeficients of polynom P. On the third line there are $n+1$ integers

b_0 ... b_n. Always a_m <> *0* and b_n <> *0*. Coeficients a_i and b_i are separated by space and they are chosen in order to each resulting coeficient could fit into the standard type *integer*.

Output Specification

The program prints exactly one line for each assignement. On this line, there will be *p+1* numbers. These numbers are coeficients c_0 ... c_p of resulting polynom. The coeficients are separated by space and the line does not consist any redundant spaces. The coeficient in the highest degree should not be zero.

Sample Input

```
3
0 0
7
-2
1 1
6 6
9 -6
3 3
-3 6 -5 1
0 3 -3 1
```

Output for the Sample Input

```
7
60 -36
-3 18 -63 123 -156 138 -86 36 -9 1
```



Osmismerka (Word Puzzle)

(**file name:** `osmismerka.c, osmismerka.p, osmismerka.C`)

What a game collection it would be without famous and well known word puzzle. Also KOKODáKH contains this game. Typical word puzzle is a rectangular array filled with letters, and the list of words. The goal is to find the words form the list in the array and to score out all their letters. After scoring out the letter can still be part of any other word (words can cross and overlap). When all words from list are found and scored out there are several remaining letters. These letters, read in rows, create the secret word. The words can be found in all eight directions, including diagonal directions.

It is clear that KOKODáKH has to bring some innovation into this game. The new rule says that the word can "jump over" the array boundary and continue on the opposite side. Upper row is thus next to the bottom row and the left collum is next to the right one. If we number the rows with numbers 0 till *R-1* and collums with numbers 0 till *S-1*, we can describe each field in the array by the coordinates. For example (0,0) is the upper left corner. Word beginning on the position (*x*,*y*) can be created by succesion of characters with coordinates (*x*,*y*), ((*x*+*i*+*R*) *mod R*, (*y*+*j*+*S*) *mod S*), ((*x*+2*i*+*R*) *mod R*, (*y*+2*j*+*S*) *mod S*) and so on. The *i* and *j* can be substitued by numbers -1, 0 and +1, but they cannot be both zero at the same time.

Input Specification

The input consists of *Z* assignments. The number of them is given by the single positive integer *Z* appearing on the first line of input. The assignements follow. Each assignement consists of array filled with letters, and the list of words which should be found in the array. The description of the array begins with the line containing two integers *R* and *S* separated by space *1 <= R,S <= 200*. *R* is the number of rows in the array, *S* is the number of collumns. The *R* lines follows, each line constist of exactly *S* lowercase letters (`a` trough `z`). The next

line is the integer *K* (*0 <= K <= 1000*) determining the number of words to be found out in the puzzle. The *K* lines follow. On each line, there is one word. The maximum length of the word is 20 characters. The word may not appear in the array and it can appear there several times. In this case all appearences are scored out. Words in the list can repeat.

Output Specification

The program prints one line for each assignement. On this line there will be all letters from word puzzle which remained in the array after scoring out all the words that appeared on the list. The letters are not separated and they are lined up in order in which they appeared in the array in row major order. At the end of the string, there is the newline character. If there are no remaining letters after scoring out all words, the program prints out empty line.

Sample Input

```
1
3 7
kroksun
pranyra
cmakrom
7
syr
kra
kroksunkrok
pranyr
rak
mak
makro
```

Output for the Sample Input

```
acm
```



Patnactka (Lloyd Fifteen Puzzle)

(**file name:** `patnactka.c, patnactka.p, patnactka.C`)

Nearly everybody knows the popular game called *Lloyd fifteen puzzle*. The puzzle consists of fifteen numbered tiles placed in the square box. The dimensions of the box are 4 x 4, one place is free. The goal is to sort the tiles by moving the neighbouring ones to the free position. In the end the tiles should be orderd by their numbers. The tile with number 1 will be in the left upper corner and next tiles are ordered in row major order. The last position (the right bottom corner) will be free.

Also KOKODáKH contains the modification of this game. The game is destinated for the advanced players so the fifteen tiles would be too little. Our game can have arbitrary dimension of the array. Your goal is to write the program which will simulate the working of this game. For a given starting situation it has to replay the succession of valid moves and to display the final state of the puzzle. The valid move is moving the tile to the neighbouring free position up, down, right or left. Any other move is not valid.

Input Specification

The input consists of *Z* assignments. The number of them is given by the single positive integer *Z* appearing on the first line of input. The assignements follow. Each assignement consists of the description of the starting state of the puzzle and the list of the moves that should be done.

At the first line of the description of the game initialposition, there are two integers *R* and *S* (*2 <= R,S <= 1000*) separated by space. *R* is a number of rows, *S* is number of columns. The

R lines of input describing the rows of the puzzle follow. On each line there is *S* integers separated by at least one and at most ten spaces. At the begining of the line can be up to ten spaces which should be ignored. There are no spaces at the end of the line. Each number is the number of the tile in the given line. First number in the first line corresponds to the upper left corner of the puzzle. All numbers are between 0 to *R.S-1* (inclusive) and each number appeares only once. Zero appear also only once and gives the position of the empty field.

The list of moves follows. The list can be empty. Each move is on the separate line and consists of one number *T* (*0 < T <= R.S-1*), which is the number of the tile which should be moved. At the end of the list there is the line containing number 0. Zero indicates the end of input and is not the part of the list of moves.

Output Specification

For each assignement the program prints out one line "`Skladacka cislo C:`" (*Puzzle number C*) where *C* is substitued by the number of the assignement. The numbering of assingement is ascending and begining with 1.

For each move in the list the program finds out if the move is valid. The move is valid if there is an empty field neighbouring. For each valid move the program prints out the following sentence at the separate line: "`Kamen T presunut KAM.`" *The tail number T moved KAM)*, where *T* is the number of the tile and *KAM* is one of the strings "`doprava`", "`doleva`", "`nahoru`" nebo "`dolu`" *(right, left, up, down)*. If the move is not valid the sentence "`Neplatny tah kamenem T.`" *(Invalid move with tile T)* is printed out.

When the list of moves ends, the program prints out the final state of the puzzle in the similar format that was used in the input. The output has to consist of *R* lines, each containing *S* numbers. The only difference from the input format is separating number by exactly one space. After each assignement (including the last one) the program prints out the empty line. Empty line consists only of the newline character.

Sample Input

```
2
4 4
 1  2  3  4
 5  6  7  8
 9 10 11 12
13 14 15  0
15
11
7
0
4 4
 1  2  3  4
 5  6  7  8
 9 10 11 12
13 14  0 15
1
0
```

Output for the Sample Input

```
Skladacka cislo 1:
Kamen 15 presunut doprava.
Kamen 11 presunut dolu.
Kamen 7 presunut dolu.
1 2 3 4
5 6 0 8
9 10 7 12
13 14 11 15

Skladacka cislo 2:
Neplatny tah kamenem 1.
1 2 3 4
5 6 7 8
```