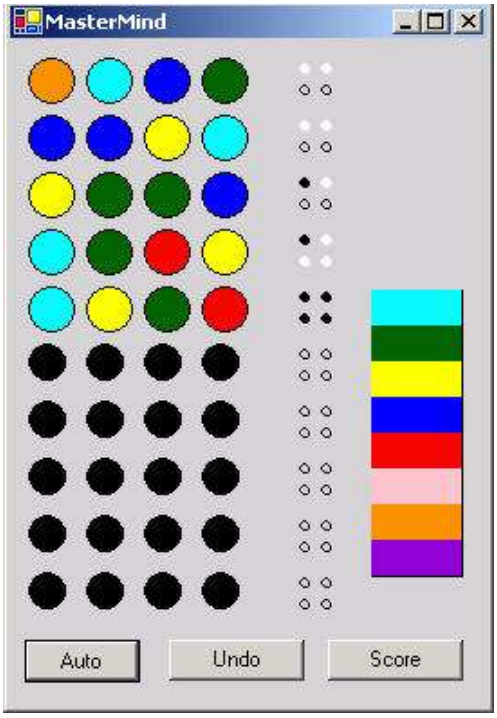
SI@UCF Example Program: Mastermind

In the popular game Mastermind, one player creates a secret code of four pegs, each of which can be chosen from one of six colors. (The number of pegs and colors may be different than this in different versions, but your implementation will use these values, but be extendible to other values. Also, a color can be reused, thus there really are six choices for each of the four pegs.) The other player then has to guess the color of each peg, with the order mattering.

The player who made the secret code then has to give feed back to the player guessing. This feedback is in the form of white and black pegs. A black peg means that the guesser has chosen the correct color in the correct slot. These are first "calculated" and awarded. Once these are counted, these pegs are ignored. Then the white pegs are awarded. These are for pegs that are the correct color but are in the incorrect slot. There is no "double dipping" of pegs in the response, so the sum total of white and black pegs the guesser can receive is four, and no one peg in the guesser's answer may earn them more than one black or one white peg.

Here is a picture of a Mastermind board:



Your task will be to analyze a partially completed game and calculate the number of possible color combinations that are completely consistent with all of the given information. Due to the fact that the manufacturer may release different versions of the game with a different number of possible colors and a different number of possible slots, your program should work where these two values are given as variables.

The Input (read from file mastermind.in)

The first line of input will contain a single positive integer, *c* (*c* ≤ *100)*, representing the number of input cases. The first line of each input case will contain three space-separated positive integers, *n*, the number of slots to fill in, *k (k* ≥ *2)*, the number of possible colors for each slot, and *m (m* ≤ *20),* the number of moves that have currently been played. It's guaranteed that $k^n \leq 10^6$. The following *m* lines will contain information about each player guess for that round. Each of these lines will start with *n* space separated non-negative integers, each in between 0 and *k-1*, inclusive, representing the player's guess, in order, for filling the slots of the secret combination. The last two integers (also space separated) on each line will be *b* and *w*, representing the number of colors in correct locations guessed correctly in the combination and the number of colors in the incorrect locations guessed correctly, respectively.

The Output (to standard out)

For each input case, output a single integer, on a line by itself, representing the number of possible combinations consistent with the partially played game for the case. You are guaranteed that each input case will have at least one possible combination meaning that the information given for the input case is correct and the person reporting the number of matches has done so.

Sample Input

```
3
4 6 4
0 1 2 3 0 2
2 2 4 1 0 2
4 3 3 2 1 1
1 3 5 4 1 3
6 10 3
0 1 2 3 4 5 0 0
7 7 8 8 9 9 0 4
9 9 7 7 8 8 4 0
10 2 1
0 1 0 1 0 1 0 1 0 1 5 4
```

Sample Output

```
1
36
200
```

Use of Global Variables

Often times, teachers discourage the use of global variables because students use them because they do not understand parameter passing. Most uses of global variables are poor, however, there are situations where they make code easier to read and debug, such as this program. The qualities of this program that make it a reasonable place to use globals are that it's fairly short, there's some common information that doesn't change while processing a single input case that is needed by several functions, and the logic and readability of the most important function (the recursive one) are enhanced if globals are used. ***Thus, for this assignment, feel free to use global variables within the context described above. No penalty will be given for doing so.***

Implementation Detail - Use of Recursion

Though this problem can be solved iteratively, the recursive solution is more elegant. ***Full credit will only be given to correct recursive solutions that use good programming style.***