Given a fraction a/b, write it as a sum of different Egyptian fraction. For example, $2/3 = 1/2 + 1/6$. There is one restriction though: there are k restricted integers that should not be used as a denominator.

For example, if we can't use 2..6, the best solution is:

$$2/3 = 1/7 + 1/9 + 1/10 + 1/12 + 1/14 + 1/15 + 1/18 + 1/28$$

The number of terms should be minimized, and then the large denominator should be minimized. If there are several solutions, the second largest denominator should be minimized etc.

Input

The first line contains the number of test cases T ($T \leq 100$). Each test case begins with three integers a, b, k ($2 \leq a < b \leq 876$, $0 \leq k \leq 5$, $\gcd(a, b) = 1$). The next line contains k different positive integers not greater than 1000.

Output

For each test case, print the optimal solution, formatted as below.

Extremely Important Notes

It's not difficult to see some inputs are harder than others. For example, these inputs are very hard input for every program I have:

```
596/829=1/2+1/5+1/54+1/4145+1/7461+1/22383
265/743=1/3+1/44+1/2972+1/4458+1/24519
181/797=1/7+1/12+1/2391+1/3188+1/5579
616/863=1/2+1/5+1/80+1/863+1/13808+1/17260
22/811=1/60+1/100+1/2433+1/20275
732/733=1/2+1/3+1/7+1/45+1/7330+1/20524+1/26388
```

However, I don't want to give up this problem due to those hard inputs, so I'd like to restrict the input to "easier" inputs only. I know that it's not a perfect problem, but it's true that you can still have fun and learn something, isn't it?

Some tips:

1. Watch out for floating-point errors if you use double to store intermediate result. We didn't use double.
2. Watch out for arithmetic overflows if you use integers to store intermediate result. We carefully checked our programs for that.

Sample Input

```
5
2 3 0
19 45 0
2 3 1 2
5 121 0
5 121 1 33
```

Sample Output

```
Case 1: 2/3=1/2+1/6
Case 2: 19/45=1/5+1/6+1/18
Case 3: 2/3=1/3+1/4+1/12
Case 4: 5/121=1/33+1/121+1/363
Case 5: 5/121=1/45+1/55+1/1089
```