2020 Denison Spring Programming Contest
Granville, Ohio
29 February, 2020

Rules:

1. There are **six** problems to be completed in **four hours**.
2. All questions require you to read the test data from standard input and write results to standard output. You cannot use files for input or output. Additional input and output specifications can be found in the General Information Sheet.
3. No whitespace should appear in the output except between printed fields.
4. All whitespace, either in input or output, will consist of exactly one blank character.
5. The allowed programming languages are C, C++, Python 2, Python 3, and Java.
6. All programs will be re-compiled prior to testing with the judges' data.
7. Non-standard libraries cannot be used in your solutions. The Standard Template Library (STL) and C++ string libraries are allowed. The standard Java API is available, except for those packages that are deemed dangerous by contest officials (e.g., that might generate a security violation).
8. The input to all problems will consist of multiple test cases.
9. Programming style is not considered in this contest. You are free to code in whatever style you prefer. Documentation is not required.
10. All communication with the judges will be handled by the PC^2 environment.
11. Judges' decisions are to be considered final. No cheating will be tolerated.

Problem A: Catan Connection

In the game *Settlers of Catan*, players start with two road segments and try to connect these up into a single long road. For this problem, you will find the fewest number of road segments that need to be added to connect the initial two road segments.

The game board is a grid of hexagons with roads segments built along the edges. A player starts with road segments on two such edges and places more road segments along the edges until they are connected. For example, in the figure below, if the player starts with road segments indicated in red, then adding just four road segments (indicated in blue) will connect them.

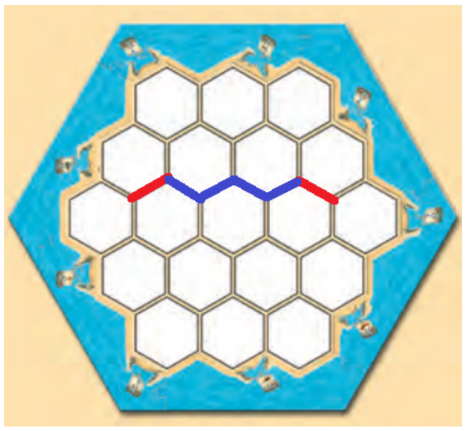

Figure 1: Catan Example 1 (image from *Catan* rulebook)

Similarly, the starting configuration below (shown again in red) needs nine road segments to be connected. One possible such path is shown in blue.

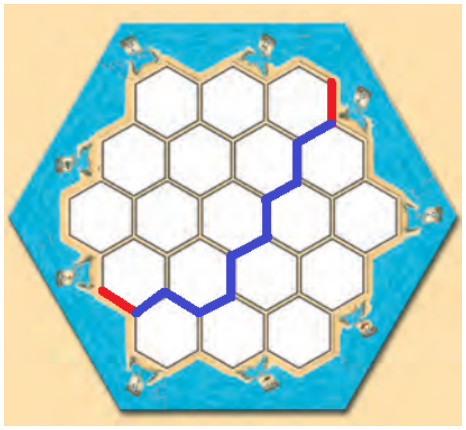

Figure 2: Catan Example 2 (image from *Catan* rulebook)

Input

Each input consists of a single line with eight integers x_1 y_1 x_2 y_2 x_3 y_3 x_4 y_4 ($0 \leq x_1, x_2, x_3, x_4 \leq 10$, $0 \leq y_1, y_2, y_3, y_4 \leq 11$). (A line with eight 0s indicates the end of input.) The first road segment will be between positions (x_1, y_1) and (x_2, y_2) and the second will be between (x_3, y_3) and (x_4, y_4), where each coordinate is given in the coordinate system in the figure below. (Note that the figure only shows the coordinates of the outer vertices—you should infer the other coordinates.) It will be the case that (x_1, y_1) and (x_2, y_2) are adjacent coordinates as are (x_3, y_3) and (x_4, y_4).

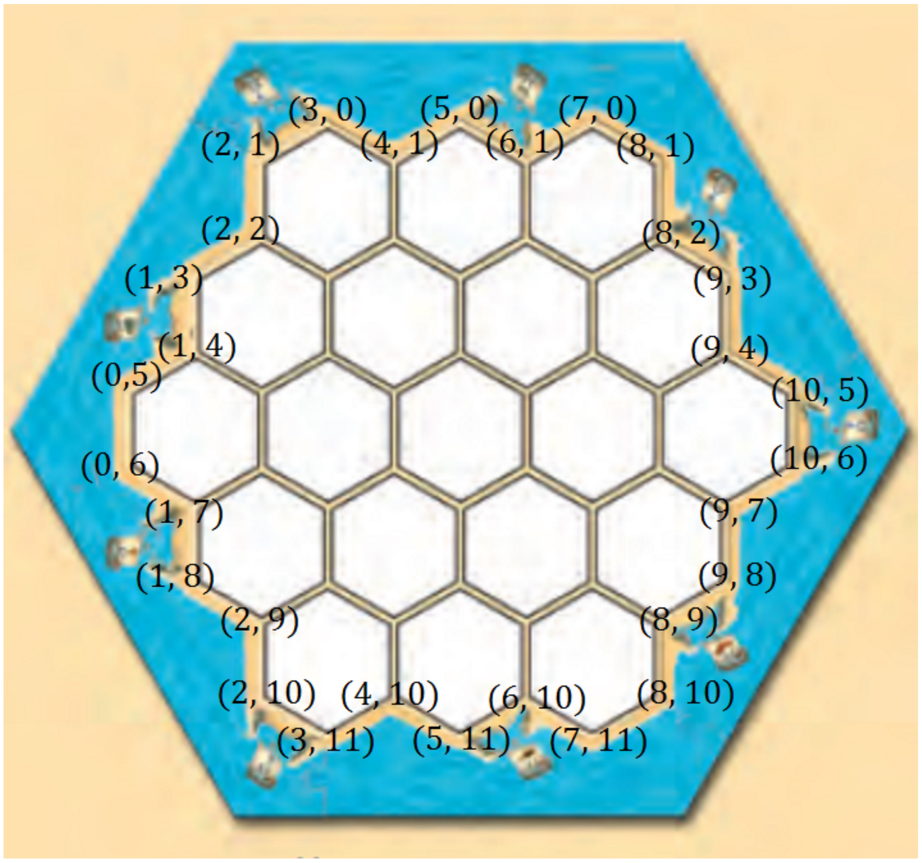

Figure 3: Coordinate system for Catan tiles (image from *Catan* rulebook)

Output

Each input set should output the case number and the minimum number of road segments needed to connect the original two segments.

Sample Input

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

Sample Output

```
Case 1: 4
Case 2: 9
```

Problem B: Fortress Feasibility

In the game *Wizards Unite*, players team up to fight foes in fortresses. Players each have one of three professions (Auror, Magizoologist, Professor) and face foes that can be one of three types (Beasts, Curiosities, Dark Forces). Each profession has a deficiency against a type of foe, a proficiency against another type of foe, and is neutral to the third. Specifically, as seen in the figure below, Magizoologists are deficient against Curiosities but proficient against Beasts; Aurors are deficient against Beasts but proficient against Dark Forces; and Professors are deficient against Dark Forces but proficient against Curiosities. The goal of this question is to figure out the fastest time a particular team of players will be able to defeat a particular collection of foes (if this is greater than the fortress timer then the players will not waste their time and resources trying to complete the challenge).

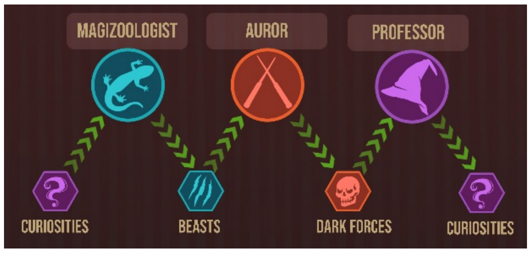

Figure 4: Profession deficiencies and proficiencies (image from *Wizards Unite*)

In a fortress, a team of up to five players will try to defeat a group of foes, each with a base time needed to be defeated. For the purposes of this problem, we will say that a player takes twice the base time to defeat a foe from a class (s)he is deficient against and half the base time (rounded down) to defeat a foe of a class (s)he is proficient against. For example, if there is a Curiosity that needs a base time of 31 seconds to be defeated, then a Magizoologist will take 62 seconds to defeat it, an Auror 31 seconds, and a Professor only 15 seconds. You have to determine whether there is a way to assign the foes to the members of a team in such a way that all players defeat all their foes before the timer runs out.

For example, suppose that there is a team with an Auror, a Magizoologist, and a Professor that is fighting the following types of foes: a Curiosity (with base time 30 seconds), three Beasts (60 seconds each), and a Dark Forces foe (20 seconds). Then the optimal assignment is for the Auror to fight the Curiosity and Dark Forces foes (30 + 10 seconds), the Magizoologist to fight two of the Beasts (30 + 30 seconds), and the Professor to fight the third Beast (60 seconds), resulting in the fortress being completed in 60 seconds (the longest completion time for any member).

Input

Each input starts with a line with four integers $F\ A\ M\ P$ where F ($1 \leq F \leq 10$) is the number of foes and A, M, P ($0 \leq A, M, P \leq 5$, $1 \leq A + M + P \leq 5$) are the number of Aurors, Magizoologists, and Professors (respectively) in the team. The next F lines will consist of the foes, one per line, in the format $T\ S$, where T ($T \in \{B, C, D\}$) is the type of foe (B=Beast, C=Curiosity, D=Dark Forces) and S ($1 \leq S \leq 100$) is the base time for the foe to be defeated by an opponent with neither a proficiency nor deficiency against that type of foe. A single line with 0 0 0 0 will terminate the input.

Output

For each input, you should output the case number and the minimum time needed for this team to defeat all foes.

Sample Input

```
5 1 1 1
C 30
B 60
B 60
B 60
D 20
0 0 0 0
```

Sample Output

```
Case 1: 60
```

Problem C: Hit The Target

You are given a list of integer values and a target integer. Your goal is to modify the list of numbers so that the sum of values in the list exactly equals the target. You can modify the list by replacing each value a_i with value f if and only if $a_i > f$. Your job is to find the smallest value f that hits the exact target.

For example, suppose the target is 50 and your list of numbers is:

9	3	19	20	1	6	4	12	18

The sum of this list is 92, larger than the target. Using a value of $f = 7.2$ the list becomes:

7.2	3	7.2	7.2	1	6	4	7.2	7.2

and the sum of the list is now exactly 50. You will report the value of f as a rational number, for instance, $f = \frac{36}{5}$ in the above example.

It may be the case that the sum of values in the list is less than the target. In this case, no such value of f exists and we should report NONE as the solution for the problem.

Input

The input is given as one problem instance per line of input. Each line begins with T and n where T ($1 \leq T \leq 10000$) is an integer that is the given target and n ($1 \leq n \leq 100$) is the number of items in the list. The line then contains n non-negative integers which are the values in the list. A value of $T = 0$ and $n = 0$ will indicate the end of input.

Output

For each input you should output the case number and the rational number as an integer numerator, a forward slash, and an integer denominator. There are no spaces between any of the three parts. The rational number f for each problem should be reduced to lowest possible terms—for example, `4/2` should be reported as `2/1`. If there is no solution, the answer should be NONE.

Sample Input

```
5 3 1 2 3
50 9 9 3 19 20 1 6 4 12 18
50 3 1 2 3
30 3 5 10 15
0 0
```

Sample Output

```
Case 1: 2/1
Case 2: 36/5
Case 3: NONE
Case 4: 15/1
```

Problem D: Leap Years

Happy Leap Day! Most people think that a leap year is any year that is divisble by four, but this is not exactly correct. More precisely, a year is a leap year if it is divisble by 4, except if it is divisble by 100 in which case it is not a leap year, except when it's divisible by 400 (that's the exception to the exception). Confused yet? A better way to think about it is to follow a simple algorithm:

- If the year is divisible by 400 it is a leap year, otherwise
- if the year is divisible by 100 it is not a leap year, otherwise
- if the year is divisible by 4 it is a leap year, otherwise
- it is not a leap year.

For example, 1996, 2000, 2020 are all leap years, but 1997, 1900, and 2100 are not leap years.

For this problem, you will design a way to apply rules of the above type to determine if a year is a leap year for different definitions of leap years. For example, if the modified leap year rule was:

- If the year is divisible by 100 it is not a leap year, otherwise
- if the year is divisible by 50 it is a leap year, otherwise
- if the year is divisible by 7 it is a leap year, otherwise
- if the year is divisible by 3 it is not a leap year, otherwise
- it is a leap year,

then in this system 350 is a leap year but 354 is not a leap year.

Input

Each input set starts with the integers y r ($1 \leq y \leq 10000$, $2 \leq r \leq 10$) on a line indicating the year and the number of rules for this case. (A line with 0 0 indicates the end of input.) The following r lines will consist of the pair L D ($L \in \{Y, N\}$, $1 \leq D \leq 1000$) where L denotes whether this rule indicates if the year is a leap year (Y) or not (N) and D indicates the divisor for this rule. The last rule will always have a divisor of 1 and will hence catch all remaining cases.

Output

For each input, you should output the case number and YES or NO indicating whether they year is a leap year in the given system.

Sample Input

```
1996 4
Y 400
N 100
Y 4
N 1
```

```
1900 4
Y 400
N 100
Y 4
N 1
350 5
N 100
Y 50
Y 7
N 3
Y 1
354 5
N 100
Y 50
Y 7
N 3
Y 1
0 0
```

Sample Output

```
Case 1: YES
Case 2: NO
Case 3: YES
Case 4: NO
```

Problem E: Measuring Cups

When cooking, many recipes call for odd amounts of an ingredient (e.g., 7/12th cup) that may not match the measuring cups that you have available to you. In this problem you will be given a target amount and a set of measuring cups of different sizes and you have to determine the fewest times you have to use cups (of any size) to measure out the target amount. For example, if you are trying to measure out 7/12 of a cup using cups of sizes $1/2, 1/3, 1/4, 1/6, 1/12$ then this is possible by using the 1/3 and 1/4 cups once each for a total of two uses. Note that it is also possible to measure 7/12 using the 1/12 cup seven times but this would be less optimal (7 vs. 2 uses).

Input

Each input will consist of a single line. The first two integers p and q will be the target fraction p/q ($1 \leq p \leq 100$, $1 \leq q \leq 100$). The next integer c ($1 \leq c \leq 10$) will be the number of measuring cups on hand. The following $2c$ integers will be the size of each of the c cups given as pairs of numerators and denominators n_i and d_i (for each $1 \leq i \leq c$, $1 \leq n_i \leq 100, d_i \in \{1, 2, 3, 4, 6, 8, 12\}$). A single line with 0 0 0 will indicate the end of input.

Output

For each input, you should output the case number and the minimum number of cup uses needed to measure out the target amount. If the target amount cannot be measured using the given cups then you should output IMPOSSIBLE.

Sample Input

```
7 12 5 1 2 1 3 1 4 1 6 1 12
3 6 5 1 2 1 3 1 4 1 6 1 12
1 24 5 1 2 1 3 1 4 1 6 1 12
0 0 0
```

Sample Output

```
Case 1: 2
Case 2: 1
Case 3: IMPOSSIBLE
```

Problem F: Nonogram

A nonogram is a puzzle, such as the one shown below, in which you have to color in a grid given the constraints that each row and column has the given continuous number of colored in squares in a row. For example, the fourth row has the sequence 2,2 as there is a run of two filled squares and then another two filled squares in that row. Similarly, the second column has a 9 since there is just a single run of nine filled squares in that column.

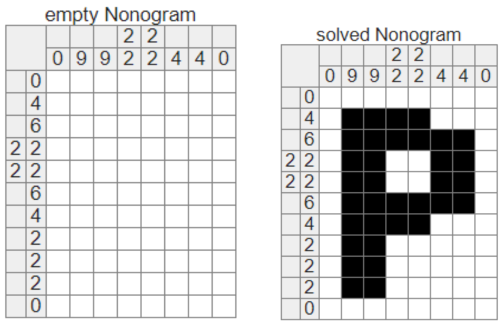

Figure 5: Unsolved and solved nonogram (taken from https://en.wikipedia.org/wiki/Nonogram)

For this problem you will construct the puzzle given a filled in grid.

Input

Each input set starts with the integers r c ($1 \leq r, c \leq 20$) on a line indicating the number of rows and columns in that puzzle. (A line with 0 0 indicates the end of input.) The following r lines will each have a string of length c with only 0s (representing empty squares) and 1s (representing filled squares).

Output

For each input, you should output the case number followed by $r + c$ lines of output. The first r lines should have the clues for each of the rows. The following c lines should have the clues for each of the columns. The clues should be output as a list of numbers separated by spaces.

Sample Input

```
11 8
00000000
01111000
01111110
01100110
01100110
01111110
01111000
01100000
01100000
01100000
00000000
3 4
1111
```

```
1001
1111
0 0
```

Sample Output

```
Case 1:
0
4
6
2 2
2 2
6
4
2
2
2
0
0
9
9
2 2
2 2
4
4
0
Case 2:
4
1 1
4
3
1 1
1 1
3
```