2024 Denison Spring Programming Contest
Granville, Ohio
24 February, 2024

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 contestant 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: FlimFlam Floats

We use the notation "d" to indicate any digit 0-9. We use "d*" to indicate any string of zero or more digits. So that "0" and "12" and "123" are all strings from d*. We use d+ to indicate any string of at least one digit (but possibly more). Thus the empty string belongs to d* but not d+.

We form a *FlimFlam Float* as a string in one of three ways:

1. a d+ string. Examples: "0", "123"
2. a d+ string, then a ".", then a d* string. Examples: "0.", "123.123"
3. a d* string, then a ".", then a d+ string Examples: "123.123", ".45"

Notice that the second and third methods produce many of the same strings. We want to compute the start sum, which is the sum of all digits that appear in the string before the ".". If there are no such digits, then the start sum is 0. We also compute an end sum which is the sum of all digits in the string appearing after the "." Again if there are no digits after the period or the period is not present, the end sum shall be 0. We will print GREATER if the start sum is greater than the end sum, LESSER if the start sum is less than the end sum, and EQUAL if they have equal value.

Input

The input begins with integer $n \geq 1$ where there are n FlimFlam Float strings. Each FlimFlam Float will be given on a separate line, making n additional lines of input. There will be at most 20 characters (including the ".") for each FlimFlam Float.

Output

For each FlimFlam Float string, you are to print the case number and either GREATER, LESSER or EQUAL. Follow the example output formatting below exactly.

Sample Input

```
5
4
.34
13.4
0.00000
123.1
```

Sample Output

```
Case 1: GREATER
Case 2: LESSER
Case 3: EQUAL
Case 4: EQUAL
Case 5: GREATER
```

Problem B: Train Parking

Edith the Engineer pulls her train into a station where there is a siding. She wants to park her train using both the siding and main line. She does so by pulling her train ahead of the siding and then letting a switcher engine (not shown) pull cars off the back of the train, putting them either into the siding or leaving them on the main line.

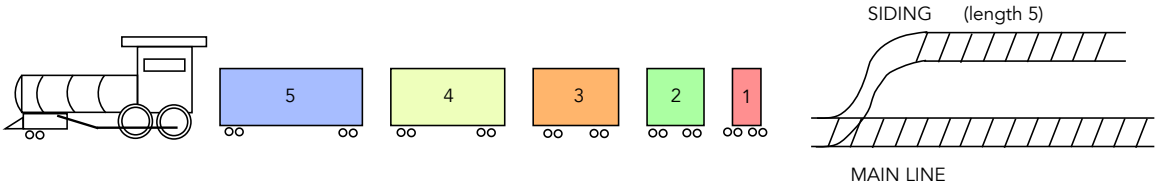

Figure 1: Train Station Parking

The siding has length k ($k = 5$ in this example). Each of the cars in Edith's train has a length, we will call them c_i for the length of car i. Edith wants to know if it is possible to park part or, possibly, all of her train cars onto the siding so that the siding is completely full. That is, the sum of the lengths of cars parked on the siding should be exactly equal to the length of the siding. In the case above, Edith can park the 5 car on the siding and completely fill up the siding. This is a good solution.

If you are sharp, you will notice that it may be possible to fill up the siding in more than one way. In this specific instance, Edith can also park the 4 and 1 cars (with total length 5) or she can park the 3 and 2 cars (also total length 5) on the siding. If there is more than one way to fill up the siding, then Edith will attempt to find the solution which minimizes the number of *cuts*.

A cut happens when the switcher engine must decouple one or more cars from the train in order to park them either on the siding or on the main track. We will also assume that Edith always parks the locomotive on the main track so if the first car in the train (the blue 5 car above) gets parked on the siding then this counts as a cut (cutting the first car from the locomotive).

- The "5" solution has two cuts. One to cut the 5-4 cars (parking 4-3-2-1 on the main track) and then a second cut to cut the locomotive from the 5 car (parking the 5 car on the siding and the locomotive on the main line).
- The "4+1" solution has 3 cuts. One to cut the 2-1. One to cut the 4-3. One to cut the 5-4.
- The "3+2" solution has 2 cuts. One to cut the 2-1. One to cut the 4-3.

Thus the "5" solution and the "3+2" solution are equally good. We are to report the minimal number of cuts that can be made to park exactly the right total length of cars on the siding. We print IMPOSSIBLE if it is not possible to find any solution where we can exactly fill up the siding with some collection of the train cars.

Input

Each problem instance, which is one problem of parking the train, is composed of integer pairs n and k where $1 <= n <= 35$ is the number of train cars and $1 <= k <= 100,000,000$ is the length of the

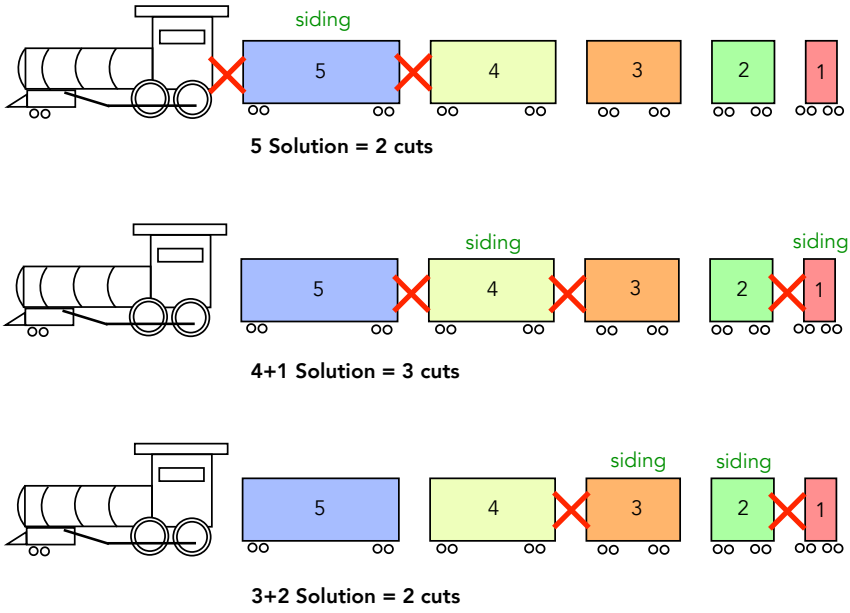

Figure 2: Solutions

siding in meters. Following these two values on the next line are n integers indicating the lengths of the n cars given as $c_1, c_2, ..., c_n$ where each $1 <= c_i <= 20$. That is, there are at most 35 train cars and each car is between 1 and 20 meters long. These values are given in the order they are located on the train where the first value (c_1) is just behind the locomotive and the last value (c_n) is at the tail end of the train (where the caboose would normally go). A problem instance of $n = 0$ and $k = 0$ indicates the end of input.

Output

For each problem, you are to print the case number and either the minimum number of cuts that can be found for filling up the siding with some collection of train cars or IMPOSSIBLE if there is no solution for parking the train cars to exactly fill up the siding. Follow the example output formatting below exactly.

Sample Input

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

Sample Output

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

Problem C: Partner Matching

Professor Snape is teaching several classes this semester with titles such as Computer Coding Wizardry, Magical Data Structures, and Analyzing the Complexity of Spells. Each class has several labs where students are grouped with lab partners. Dr. Snape wants to be sure that for class this semester no two students end up in the same lab group more than once.

For instance, we consider his Analyzing the Complexity of Spells class with 7 students named A, B, C, D, E, F and G. Snape has constructed groupings for the three labs this semester as shown below:

Lab	Groups
1	(A,B,C) (D,E) (F,G)
2	(A,D,F) (B,E), (C,G)
3	(A,F) (B,G) (C,E) (D)

Figure 3: Snape's Attempt at Lab Grouping

This grouping is almost perfect except that (A,F) are partners in both the second and third labs. So this is not a viable solution. Notice that Snape uses non-uniform group formations. Each lab can have a different number of groups and there can be different number of students in each group, including groups of just one student.

Input

Each problem instance, which is one of Snape's classes for the semester, is given by a pair of integers, n and m where $1 \leq n \leq 26$ is the number of students and $1 \leq m \leq 20$ is the number of labs for that class. The students will be named starting with A, then B and so on for up to 26 students (thus named Z) for the problem instance. Following the n and m pair will be m lines of input where each line is a lab grouping of the students. The syntax of the grouping will be as shown in the example input (below):

- Each grouping line begins with integer k and is followed by k different groups.
- The groups for each lab will be separated by a single space.
- There will be no other whitespace anywhere in that line (that grouping).
- Each group will start with "(" and end with ")".
- The students in each group will be separated by commas.
- No group will be empty; they will all have at least one student.
- All students will appear exactly once in a grouping (there will be no duplicates of students and there will be no missing students). Formally, a grouping is a partition over the set of students.

A value of $n = 0$ and $m = 0$ indicates the end of input.

Output

For each problem, you are to print the case number and either YES if the group attempt for that class is valid or NO if it is not. A set of groupings is valid if and only if no two students end up as partners in multiple labs. Follow the example output formatting below exactly.

Sample Input

```
7 3
3 (A,B,C) (D,E) (F,G)
3 (A,D,F) (B,E) (C,G)
4 (A,F) (B,G) (C,E) (D)
5 2
2 (A,B,C) (D,E)
3 (A,D) (B,E) (C)
0 0
```

Sample Output

```
Case 1: NO
Case 2: YES
```

Problem D: Final Exam Scheduling

Professor Homer Simpson is teaching an introductory computer science course this semester. His students have taken 7 quizzes during the semester. For the final exam, Dr. Simpson will allow his students to retake any 3 of the quizzes so they can try to raise their quiz grade average. Homer realizes that there are 4 final exam time slots he can use. He would like to schedule each of the 7 quiz retakes in one of the final exam time slots. But he is not sure if it is possible so that no student needs to take 2 different quiz retakes during the same exam time slot.

For example, consider the table below showing seven of Homer's students. The "x" marks a quiz that each student wants to retake for the final.

Student	**Quiz1**	**Quiz2**	**Quiz3**	**Quiz4**	**Quiz5**	**Quiz6**	**Quiz7**
Alice		x				x	x
Bob			x			x	x
Carol			x	x			x
Dave	x		x		x		
Eve			x		x		x
Fred		x			x		x
Glenda			x	x		x	
Henry					x	x	x
Indigo	x		x	x			
Juan				x			

Figure 4: Quiz Retake Chart

Notice that each student can take up to three quiz retakes, but they can take fewer.

In this example, Homer finally figures out he can schedule the quizzes as follows:

Exam Slot	**Quiz Retakes**
1	Quiz1, Quiz 6
2	Quiz2, Quiz 3
3	Quiz4, Quiz5
4	Quiz7

Figure 5: Quiz Schedule

There are no conflicts where a student wants to take two different quizzes during the same final exam time slot. Homer is happy. Homer's students are even happier.

Input

Each problem instance, which is one instance of Homer's class, is given by four integers (n, q, r, f) on the first line:

n The number of students in the class who are retaking quizzes.

q The number of quizzes given during the semester.

r The maximum number of quizzes each student can choose to retake for the final.

f The maximum number of final exam time slots.

Following this are n lines of input where each line contains up to r integers (there will be at least 1 integer, up to r). Each line represents a student and the integers are all quiz numbers that the student wishes to retake.

A value of $n = 0, q = 0, r = 0, f = 0$ indicates the end of input. The bounds for each variable are:

n 1 to 10

q 1 to 7

r 1 to 4 with $r \leq q$

f 1 to 4

Output

For each problem, you are to print the case number and either YES if it is possible to schedule the q quiz retakes using at most f different time slots so that there are no conflicts. Otherwise you print NO. Follow the example output formatting below exactly.

Sample Input

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

Sample Output

```
Case 1: YES
Case 2: NO
```

Problem E: Crash Test Sentences

The NHTSA (National Highway Traffic Safety Administration) uses crash test dummies in cars and then smashes cars into fixed barriers to see what happens to the car's occupants. These are important safety tests that promote the design of safe automobiles. But most people don't know that the NHTSA also crash tests sentences too!

Figure 6: Crash Testing a Sentence

When a sentence is revved up to high speed and smashed into a barrier, a couple of things happen to the sentence.

- Any non-letters get obliterated. Spaces, punctuation and other unfortunate non-letter characters simply vanish in a puff of grammatical dust. [1]
- Vowels absorb the brunt of the impact. When they are smashed at high speed they change shape. A's become E's, E's transform to I's, I's become O's, O's change to U's and finally U's change to A's. The same thing happens with lower case vowels too (a to e, ...).
- Vowels also have an effect on the character that immediately comes behind them. If the character following a vowel is a letter (a-z, A-Z) then they change case. Uppercase becomes lowercase and vice-versa.

Note: all three actions happen simultaneously. We do not apply them in order nor do the effects of one step influence the actions of another step.

[1] It is conjectured that such dust is actually comprised of theoretical sub-grammatical particles such as noun-ons, verb-ons, adverb-ons and so on. The scientists at the CERN super collider are experimenting to prove such things exist.

Consider the sentence "Hello, World!". When this collides at high speed with a fixed object it transforms into the sentence"HiLluWuRld". Poor, poor sentence. But at least we know that its sacrifice helps keep other sentences safe in the unlikely event of high speed collisions of our books and research papers.

Input

The input begins with integer $1 \leq n \leq 30$. Following are n sentences, one per line. Each sentence is to be smashed into a crash test wall at high speed. Sentences will contain at most 100 characters (including punctuation and space).

Output

For each problem, you are to print the case number and then the resulting sentence (after the crash). Follow the example output formatting below exactly.

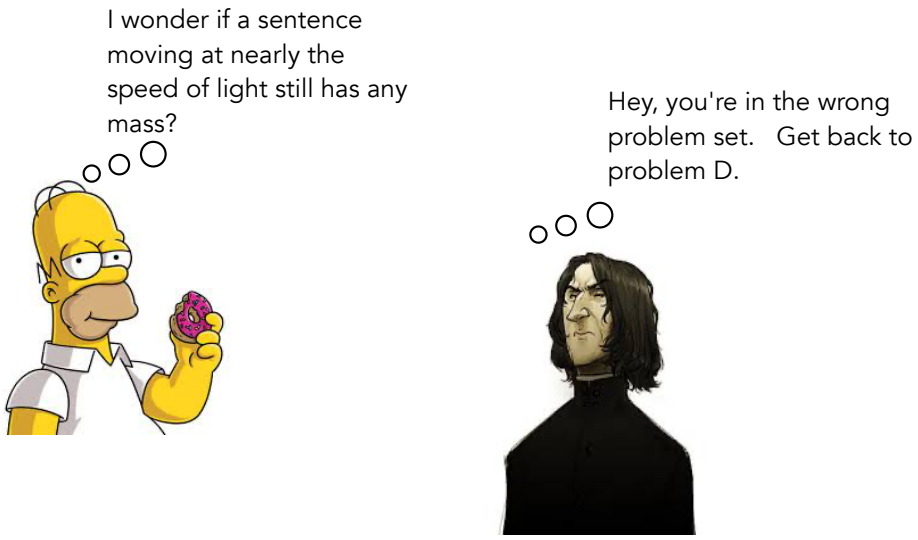

Figure 7: Deep Thoughts

Sample Input

```
2
Hello, World!
Gravity cannot be held responsible for people falling in love.  -- Albert Einstein
```

Sample Output

```
Case 1: HiLluWuRld
Case 2: GreVoTyceNnuTbihiLdriSpuNsoBlifuRpiUPlifeLloNgoNluViELbiRtIONstiON
```

Problem F: Stacking Boxes

Nguyen has his first job out of college. Today he is stacking a series of boxes into several stacks. For example, let's assume we have two stacks which we will call Stack A and Stack B. Each box has the same length and width, but they come in a variety of different heights. Nguyen follows a very simple rule for stacking the next box:

> For each box, Nguyen puts it on top of the shortest stack. If more than one stack has the same shortest height, then Nguyen puts the new box on the shortest stack that is nearest to him (Stack A is the nearest, B is next nearest, then C, ...).

For example, if Nguyen has these two stacks and he receives boxes of heights 5, 12, 4, 2, 9, 8, and 6 in that order, then the stacking will be:

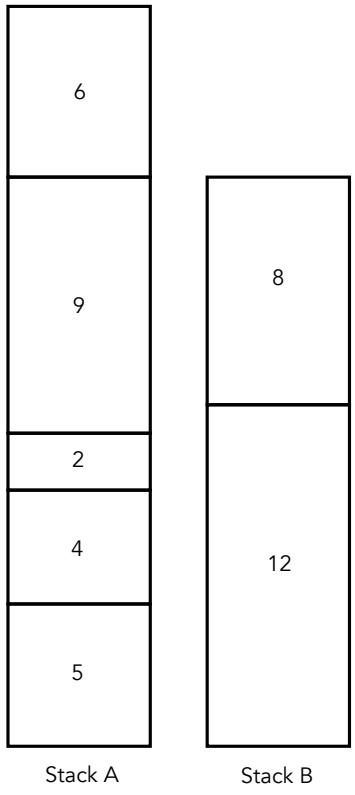

Figure 8: Stacking Boxes

Finally Nguyen's boss, Alice, comes and asks him to retrieve a specific box – in this case the box of height 2. Nguyen realizes he must move two boxes on top (the 6 and 9 boxes) in order to reach the 2 box. For this problem, we want to know the minimum number of boxes we must move in order to reach the box specified by Alice.

Input

Each problem instance, which is a box stacking problem as described above, is given by a triple of integers, n, m and k where $1 \leq n \leq 20$ is the number of boxes, $1 \leq m \leq n$ is the number of stacks,

and $1 \leq k \leq 100$ is the specific box height that Nguyen's boss will ask him to retrieve later. The next line contains the n box heights in the order they are given to Nguyen: $b_1, b_2, ...b_n$ where each box is $1 \leq b_i \leq 100$ units high. At least one of those box heights will have value k.

A value of $n = 0$ and $k = 0$ indicates the end of input.

Output

For each problem instance, you are to print the case number and then the minimum number of boxes that must be removed from some stack in order to reach a box of height k. If there are multiple box heights of value k, then you are to find the box of height k that is most accessible – with the fewest number of boxes on top. Follow the example output formatting below exactly.

Sample Input

```
7 2 2
5 12 4 2 9 8 6
7 2 4
3 8 4 2 4 2 2
8 3 5
1 2 3 5 4 6 10 1
0 0 0
```

Sample Output

```
Case 1: 2
Case 2: 0
Case 3: 1
```