University Interscholastic League

Computer Science Competition

2015 State Programming Problem Set

DO NOT OPEN THIS PACKET UNTIL INSTRUCTED TO BEGIN!

I. General Notes

1. Do the problems in any order you like. They do not have to be done in order from 1 to 12.

2. All problems have a value of 60 points. Incorrect submissions receive a deduction of 5 points, but may be reworked and resubmitted. Deductions are only included in the team score for problems that are ultimately solved correctly.

3. There is no extraneous input. All input is exactly as specified in the problem. Unless specified by the problem, integer inputs will not have leading zeros. Unless otherwise specified, your program should read to the end of file.

4. Your program should not print extraneous output. Follow the form exactly as given in the problem.

II. Table of Contents

1. Bully

Program Name: Bully.java   Input File: bully.dat

There is a bully in school named Castanon. Every day you bring a Beetleborg toy to school he smashes it to pieces. But thanks to the advances of modern technology, this brand of toy is made of interchangeable parts and can be reconstructed from broken parts. Sometimes, the damage is so great that the toy cannot be put together again from the original pieces. However, there is a special property of this toy, a single Beetleborg can be constructed from `R` destroyed Beetleborgs. So you can always build a Beetleborg as long as you have enough parts, and you are determined to bring a Beetleborg to school each day as long as you have a Beetleborg at home. Castanon is equally determined to destroy the one Beetleborg that you bring until you run out of Beetleborgs.

Input

The first line of input contains `T`, the number of test cases that follow. The next line contains a number `B`, the number of Beetleborgs you start with, followed by a space, and a number `R`, the number of destroyed Beetleborgs it takes to build a new one.

Output

For each test case, print the number of Beetleborgs destroyed.

Constraints

$1 <= T <= 100$
$1 <= B <= 1x10^8$
$1 <= R <= 1x10^3$

Example Input File

```
3
3 2
2 2
5 4
```

Example Output to Screen

```
5
3
6
```

2. Compression

Program Name: Compression.java   Input File: compression.dat

Rachel has been given a text file by her professor with the task of compressing the contents without losing any data, a lossless algorithm. She has been informed in advance that the text will contain only alphanumeric characters. For example:

aaaa56666bmFfff

She has decided on using the following compression algorithm: Count the number of repeating characters, and place it before that character. The output for the above input would be

4a15461b1m1F3f

Your job is to produce the output according to Rachel's algorithm.

Input
The first line of input contains T, the number of test cases that follow. The subsequent lines will each hold a single test case where the length of the test case is N.

Output
For each test case, print the result of the compression process.
Note: It may be the case that the compression results in a lengthier string than the original.

Constraints
$1 <= T <= 100$
$0 <= N <= 1x10^3$

Example Input File
```
3
aaaa56666bmFfff
12121212
CCCCCCCssssssss
```

Example Output to Screen
```
4a15461b1m1F3f
1112111211121112
7C8s
```

3. Decisions

Program Name: Decisions.java Input File: decisions.dat

You are a strategic planner of routes. You need to deliver a bunch of goods to a relief zone from various warehouses. All of your warehouses have the appropriate goods, but the catch is, once you choose a warehouse, you cannot select any other warehouse to deliver the goods. As a result, you would like to make the trip from the warehouse to the relief zone as short as possible while only being able to travel up, down, left and right.

Input
The first line contains an integer, T, to denote the number of test cases. In each test case, the first line contains two integers, R and C, to denote the number of rows and columns in the map. For the following R lines, each line contains C characters for the map. Every character in the map will be one of the following characters:

. (Period) – You can travel on this space
# (Pound) - You cannot travel on this space
W (Capital W) – These are the locations of your warehouses
R (Capital R) – This is the relief zone you need to get to

Output
Please print out the row and column of the closest warehouse. If there is a tie, print out the row and column of the warehouse with the smallest row. If there is still a tie then, print out the row and column of the warehouse with the smallest column.

If there is no path to the relief zone, then print **Send a helicopter**.

Constraints
1 <= T <= 10
2 <= R <= 100
2 <= C <= 100
1 <= Number of warehouses <= 50

Example Input File
```
3
7 7
#W#.R##
...#.##
...#...
.##..W.
.#....W
.##.##.
.......
5 5
W.W.W
.....
W.R.W
.....
W.W.W
3 10
.W.###....
....###...
...###...R
```

Example Output to Screen
```
3 5
0 2
Send a helicopter
```

4. Land

Program Name: Land.java   Input File: land.dat

According to scientists, the water on our earth is running out. Soon, there will only be a single continent left that has any water available. The scientists are not sure how long we have before this occurs, so they have decided to leave the job up to you to perform the calculation.

It is year WXYZ, and the scientists have mapped the surface of the earth out onto an NxM grid of integers. The surface of the earth is either land or water. Each positive integer represents a patch of surface covered by water, with the value of the integer representing how many days from now until the water runs out. Once those days are over, it becomes a patch of land. Patches of land are represented by zeroes. Since we are living on land, this grid is guaranteed to have a 0 somewhere.

Given this NxM grid representation of the earth, compute how many days it will take for Earth to be united into a single patch of land. Two patches of land are connected if they're above/below or to the left/right of one another. Assume that the Earth is now rectangular and there is no wrap-around.

Input

The first line of input will contain a single integer T, the number of test cases.
The first line of each test case contain two integers, N and M, the number of rows and the number of columns in the grid respectively.
The following N lines of input will contain M nonnegative integers. No integer will exceed the value, K.

Output

For each test case, print out the minimum number of days we have left until all patches of land are connected as one.

Constraints

```
1 <= T <= 10
1 <= N <= 1000
1 <= M <= 1000
0 <= K <= 10^9
```

Example Input File

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

Example Output to Screen

```
1
9
0
```

5. Receipt

Program Name: Receipt.java   Input File: receipt.dat

Your grocery store is going out of business due to the arrival of a new competitor, the HED Corporation's Family Mart! After a failed effort to infiltrate their store, you are banned from entering the premises of any Family Mart. However, your assistant goes through the trash bin at each HED location and can retrieve receipts. Since HED does not want its customers to know how much each item is worth, the receipts only contain the number of items bought and a total price.

In order to remain competitive, you need to adjust your prices to match HED's. However, your dim-witted assistant only has two hands and thus only brings you two receipts from each HED. Can you use the receipts to figure out the price of each item?

Input

The file begins with an integer `T (1 ≤ T ≤ 10)`. After that, `T` test cases follow. Each test case consists of exactly 3 lines. The first line contains two space-separated words, the names of the items bought. The second line contains two integers and a double, which represents the number of the first item purchased, the number of the second item purchased, and the total price of the purchase respectively. The quantities will fit in a 32-bit integer, and the total price will fit into a double. The third line is another receipt in the same format.

Output

For each HED store, print a line "`Grocery Store #N:`", where `N` is the index of the grocery store in the input. If it's possible to reconstruct the prices of the original goods, print the answer in the following syntax: "`item1: cost1, item2: cost2`". If it is impossible to compute the prices of the goods, print the line "`CANNOT COMPUTE PRICES`" instead.

Example Input File

```
3
apple orange
1 2 6.00
2 4 12.00
eraser pencil
1 2 2.50
3 1 2.50
bread butter
2 3 5.00
10 16 26.00
```

Example Output to Screen

```
Grocery Store #1:
CANNOT COMPUTE PRICES
Grocery Store #2:
eraser: 0.50, pencil: 1.00
Grocery Store #3:
bread: 1.00, butter: 1.00
```

6. Scheduler

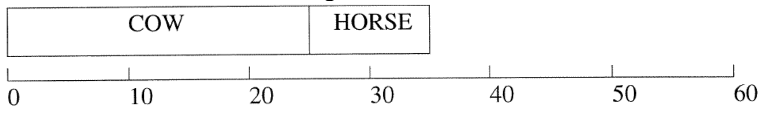
Program Name: Scheduler.java Input File: scheduler.dat

Real-time systems are systems that are subject to a “real-time constraint”. Essentially, real-time programs must guarantee response within specified time constraints, or “deadlines”. Periodic real-time systems are ones in which the programs have periodic tasks. If a set of periodic tasks can be scheduled in such a way that their constraints are always met, then that set of tasks is called “schedulable”.

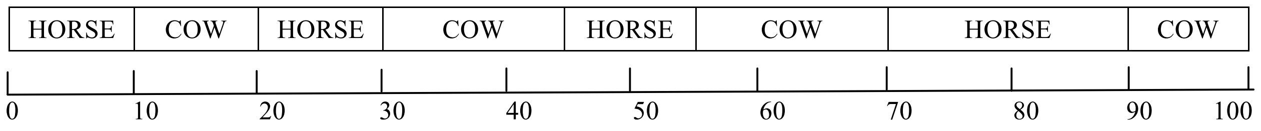
Given the periodic requirements, it is trivial to show that the tasks are schedulable if and only if there exists a valid scheduling for the first N minutes, where N is the LCM (least common multiple) of the periods.

For an example of a periodic system, consider one in which you are a rancher and you must feed your horses for at least 10 minutes every 20 minutes, and your cattle for at least 25 minutes every 50 minutes. The period for the horse is 20 minutes, and the execution time is 10 minutes. The period for the cattle is 50 minutes, and the execution time is 25 minutes. The deadline for each matches the period. They eat in the same barn out of the same troughs so they cannot ever eat at the same time. Assume the required times include the time it takes to change from feeding horses to cows and back each time.

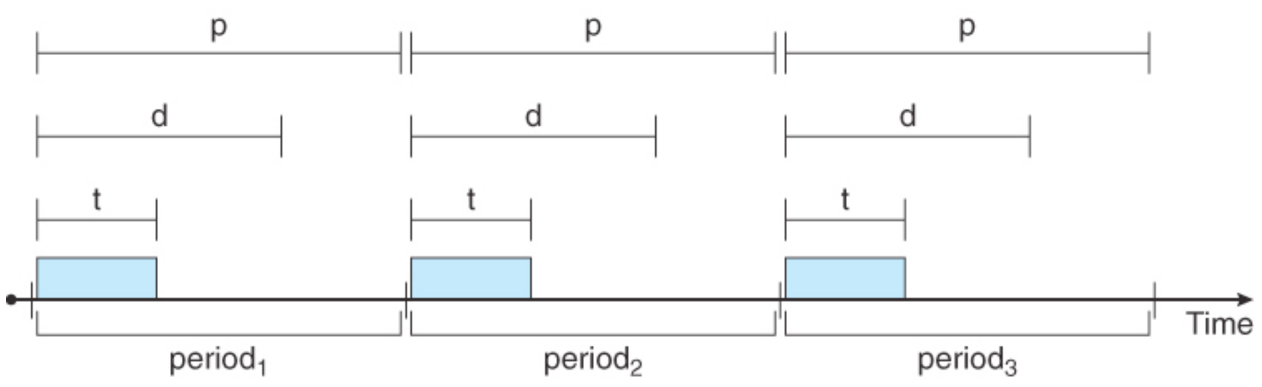
If you take turns feeding the cattle and then the horses, then the horses would starve to death before they are fed and so this is not a valid scheduling order:



If, on the other hand, you were to feed them in the following order, then it is possible to satisfy the given constraints for the full 100 minutes. Note that the LCM of 20 and 50 is 100. For this example, both the horse and the cow had deadlines that corresponded to their period. That is, as long as the horse and the cow were fed for 10 and 25 minutes every 20 and 50 minutes, respectively, then everything was fine. Note the last 30 minutes below could also have been Horse - 10, Cow - 10, Horse - 10 and still met the given constraints.



Not all tasks have deadlines that correspond to their periods though. In this next example, a user might click a button every 60 seconds expecting an on-screen response within 3 seconds of the click, but the response may take 1 second for the computer to compute. In this case, from the moment the button is clicked to start the task until it is clicked again to start the next task, the period is 60 seconds, the deadline is 3 seconds, and the execution time is 1 second.



click **click** **click**

Input

The first line contains a single integer N, the number of test cases to follow. The first line of each test case contains a single integer M, the number of tasks to schedule. The following N lines each represent a single task, made up of three integers in the form P D T, where P is the period with which the task repeats, D is the deadline with which the task must be completed each time it repeats, and T is the time required to execute the task. For instance, a button which will be clicked every 60 seconds, expecting an on-screen response within 3 seconds, and the response taking 1 second to compute, would be written as '60 3 1'.

Output
For each set of periodic tasks, determine if there exists some scheduling order such that the constraints can always be met. Print SCHEDULABLE if some order exists, or print NOT SCHEDULABLE if no order exists.

Constraints
1 <= N <= 10
1 <= M <= 8
1 <= LCM <= 1024
1 <= T <= D <= P <= 1024 for all tasks

Sample Input File
```
4
2
20 20 10
50 50 25
2
3 3 1
4 4 2
2
2 1 1
2 2 1
2
2 1 1
2 1 1
```

Example Output to Screen
```
SCHEDULABLE
SCHEDULABLE
SCHEDULABLE
NOT SCHEDULABLE
```

Explanation of Output

In the first set of tasks, there are two tasks corresponding to the rancher example above. Since, as per the diagrams, we know that there is a solution, the result is SCHEDULABLE.

In the second set of tasks, there are two tasks. The first task appears every 3 seconds, and within 3 seconds of appearing must be executed for at least 1 second. The second task appears every 4 seconds, and within 4 seconds of appearing must be executed for at least 2 seconds. The LCM of 4 and 3 is 12. The following is a valid scheduling of these two tasks:
Time 0 – 1: Task 1
Time 1 – 3: Task 2
Time 3 – 4: Task 1
Time 4 – 6: Task 2
Time 6 – 7: Task 1
Time 7 – 8: Task 1 or 2
Time 8 – 10: Task 2
Time 10 – 11: Task 1
Time 11 – 12: Task 1 or 2

In the third set of tasks, there are two tasks. The first task appears every 2 seconds, and within 1 second of appearing must be executed for at least 1 second. The second task appears every 2 seconds, and within 2 seconds of appearing

must be executed for at least 1 second. The LCM of 2 and 2 is 2. The following is a valid scheduling of these two tasks:
Time 0 – 1: Task 1
Time 1 – 2: Task 2

The fourth set of tasks is not schedulable. Both task 1 and task 2 must be executed for at least 1 second, within 1 second of appearing. Since both tasks appear at the beginning and we can only execute one of them at a time for one second, the other task will be overdue.

7. Scoreboard

Program Name: Scoreboard.java **Input File: scoreboard.dat**

Since you are currently competing in UIL CS, you are already aware of how the scoring procedure works. The programming portion consists of 12 programming questions, each worth 60 points. A problem is considered solved by a team if any of that team's submission for the problem is accepted. A penalty score of 5 points is deducted for each incorrect submission before an accepted submission. This means that if a team submits multiple incorrect solutions for a problem but never correctly solves it, they are not penalized. Whereas a team that submits a problem correctly on the second attempt only receives 55 points. The number of points that a team receives for correctly solving a problem should never drop below 0.

Input

The first line of input contains T, the number of test cases that follow. This is followed by a blank line. There is also a blank line between consecutive test case inputs. Each test case represents a snapshot of the judging queue, containing entries for some or all of the teams 1 through 25 solving problems 1 through 12.
The first line of each test case contains N, the number of entries in the judging queue. The following N lines of input each represent a single entry. Each entry is in the format: team number, problem number, and the *team problem status*, where the *status* can be `A`, `I`, `C`, `U`, `E`. These symbols stand for Accepted submission, Incorrect submission, Clarification request, Unjudged submission, and Erroneous submission. Clarification requests, unjudged submissions, and erroneous submissions do not affect the scoring.

Output

The output for each test case will be the scoreboard, sorted by the total points earned using the criteria described above. Each line of output will contain the team number, the number of problems solved by that team, and the total score accumulated by the team. Since not all teams may be actively participating, only display the teams that have made submissions (accepted, incorrect, unjudged, and erroneous). If two or more teams are tied, they are displayed in order of increasing team numbers.

The output of two consecutive test cases will be separated by a blank line.

Constraints

```
1 <= T <= 10
1 <= N <= 25
```

Sample Input File

```
3

5
1 2 I
3 1 A
1 2 C
1 2 A
1 1 A

6
5 2 U
5 1 C
14 0 C
14 4 E
5 0 C
2 1 C

9
21 3 I
15 2 A
21 3 C
2 1 A
21 12 A
21 7 U
21 3 A
15 5 E
6 1 E
```

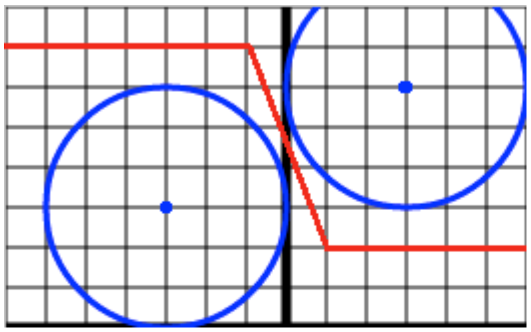
Example Output to Screen

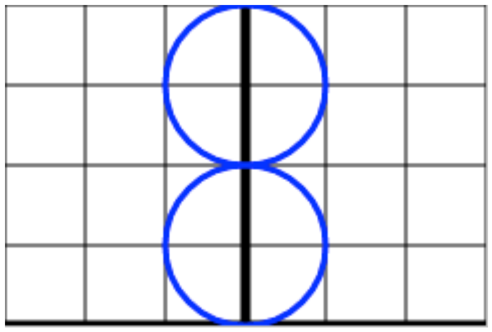
```
1 2 115
3 1 60

5 2 115
14 1 60

21 2 115
2 1 60
15 1 60
6 1 0
```

8. Siren

Program Name: Siren.java   Input File: siren.dat

In ancient Greece, there were mythical creatures called Sirens that attracted sailors by singing, and caused them to shipwreck on their island. The Greek astronomer and mathematician Eratosthenes is sailing from one side of the sea to the other. He has predetermined the location of the island for each Siren, and the range of each of their songs so that he might be able to avoid them. If his ship passes in the range of an island of Sirens, he will become attracted and crash into their island. You must write a program that given the width of the sea and locations and ranges of the Sirens, determine whether Eratosthenes can reach the other side or not.
We will define the sea as a coordinate system. Let the width of the sea be W. The coordinate system is from negative infinity to infinity on the x axis, and 0 (top) to W (bottom) on the y axis. Sirens will always be on an island so their y axis will always be 1 <= y <= W-1. Assume Eratosthenes starts at negative infinity and wants to move to positive infinity (at any y value). Each Siren is defined by an x value, y value, and radius.
Left: W = 8, Siren at (-3, 3) with radius 3 and Siren at (3, 6) with radius 3: A possible route for Eratosthenes is in red.
Right: W = 4, Siren at (0, 1) with radius 1 and Siren at (0, 3) with radius 1: No possible route for Eratosthenes.





Input
The first line will contain the number of test cases T.
The first line of each test case contains two integers: N and W, which are the number of Sirens on the map and the width of the sea.
The next N lines for each test case consist of each Siren, one per line, in the form of three integers X Y R, where X and Y are the coordinates as specified above, and R is the radius of the Siren's song.

Output
For each test case, output **YES** if Eratosthenes can make it from one side to the other safely, otherwise output **NO**.

Constraints
1 <= T <= 10
0 <= N <= 30, 0 <= W <= 200
-1000 <= X <= 1000, 1 <= Y <= W-1, 0 <= R <= W

Example Input File

```
3
2 8
-3 3 3
3 6 3
2 4
0 1 1
0 3 1
3 14
-22 3 5
-10 4 8
7 7 7
```

Example Output to Screen

```
YES
NO
YES
```

Explanation of the example
In the first case, there is space between the two circles for Eratosthenes to move through without hearing either of the Siren's calls, so the answer is YES. In the second case, the Siren's circles touch, and span the entire width of the channel, so there is no possible way for Eratosthenes to make it to the other side, so the answer is NO. In the third sample, if you visually construct the graph, it is again clear there is a path to cross the sea.

9. Speedy

Program Name: Speedy.java   Input File: speedy.dat

There are N people in a race. Some people have slow reaction times, and thus they start the race at different times. Because they all train differently, they all have different max running speeds. When the participants start the race, they immediately reach their max running speed. In the nature of the racing spirit, every time someone passes another person, the trailing person yells “Hey!” in hopes of slowing down the other (this effort has no effect). You as an observer of the race want to determine how many “Hey!”s have been yelled at a given time, T.

Input
The first line contains an integer C to denote the number of test cases. For the following C test cases, each test case starts with two integers, `N T`. The following N lines, each line follows the format of `start_time speed` to denote the start time and speed of the racer.

Output
Print out the number of “Hey!”s that have been yelled at time T.

Constraints
```
1 <= C <= 10
1 <= N <= 20
10 <= T <= 10,000
0 <= start_time <= 10,000
1 <= speed <= 10,000
start_time <= T
```

Example Input File
```
3
4 10
0 4
1 2
3 10
2 3
4 100
0 50
1 25
2 12
3 6
8 10000
9564 355
634 8324
1512 133
3524 959
1289 735
846 4604
5115 8283
9753 9216
```

Example Output to Screen
```
4
0
6
```

10. Spreadsheet Columns

Program Name: SpreadsheetColumns.java **Input File: spreadsheetcolumns.dat**

You have been assigned the task of entering data into a spreadsheet. You know the speed at which you can enter data into a single cell so you have written an equation to calculate the number of minutes it will take you to finish your work by multiplying the time it takes to input data into a single cell by the number of cells, ignoring time it takes to switch among cells. You are given the number of rows and columns which you must enter data for the entire spreadsheet. For example, if there are 5 rows and 4 columns, you will be entering data into 20 cells. If there are 9 rows and 3 columns, you will be entering data into 27 cells. The catch is the rows and columns will be given as letters. This is the system by which letters are mapped to integer values:

A = 1, B = 2, Z = 26, AA = 27, AB = 28, AD = 30, AAA = 703

Input
The first line of input contains `T`, the number of test cases that follow. The next line contains a number `M`, the number of minutes it takes you to input data into a single cell, a string or integer `R` which gives you a clue for the number of rows, and a string or integer `C` which gives you a clue for the number of columns.

Output
For each test case, on a separate line, print the number of minutes it will take for you to finish your task. It is guaranteed that the number of minutes will not overflow a signed 32-bit integer.

Constraints
```
1 <= T <= 100
1 <= M <= 10,000
1 <= R <= 2,000,000,000
1 <= C <= 2,000,000,000
```

Example Input File
```
3
4 C C
5 AA AAA
3 5 AA
```
Example Output to Screen
```
36
94905
405
```

11. Square

Program Name: Square.java   Input File: square.dat

When you start to learn about computer vision, you start discovering squares in all images. In this problem you are going to determine the number of squares in an image. You shall be given an image of size NxM where each pixel is either white or black. For the sake of the problem, we'll say that '.' is white and 'x' is black.

Determine how many different fully black squares exist in the images.

Input

The first line of input will contain a single integer T, the number of test cases.
The first line of each test case contains two integers, N and M, the number of rows and columns respectively
The following N lines each contain a M characters, which will be either a '.' or 'x'.

Output

For each test case, print out the number of fully black squares there are in the image.

Constraints

```
1 <= T <= 10
1 <= N <= 100
1 <= M <= 100
```

Example Input File

```
3
2 2
xx
.x
2 2
xx
xx
3 3
xx.
xxx
.xx
```

Example Output to Screen

```
3
5
9
```

12. Word Search

Program Name: WordSearch.java   Input File: wordsearch.dat

Everyone loves word search puzzles. You are given a rectangular NxM grid of characters and are asked to find a collection of words. However, in this case, you have been given a more challenging task. Your word search has been expanded infinitely. Your typical NxM grid of characters has a replica above, below, left and to the right of itself. All of those grids also have replicas.

So, for example let's say your grid of characters is:

```
A B

C D
```

After the first expansion, it will look like this:

```
   A B

   C D

 A B A B A B

 C D C D C D

   A B

   C D
```

After the second expansion, it will look like this:

```
        A B

        C D

      A B A B A B

      C D C D C D

    A B A B A B A B A B

    C D C D C D C D C D

      A B A B A B

      C D C D C D

        A B

        C D
```

After the second expansion completes, all of the newly added grids will then also expand to make an even larger grid. This continues on infinitely. Given a collection of words, determine which of those words exist in the grid going vertically, horizontally, or diagonally at any location. The words can also be in backwards order, just like any typical word search puzzle.

Input

The first line of input will contain a single integer T, the number of test cases.
The first line of each test case contain three integers N, M, and W: the rows of the original grid, the columns of the original grid, and the number of words you need to look for respectively.
The following N lines will each contain M space separated uppercase letters.
The following W lines will each contain a string of only uppercase characters. No string will exceed a length of K.

Output

For each test case, print out a binary string (a string of only 0's and 1's) where a 0 indicates the corresponding word in the list does not exist in the grid and 1 says it does. So if the first two words can be found in the infinite word search, but the third word cannot be found, then you would output "110" without the quotations.

Constraints

```
1 <= T <= 10
1 <= N <= 100
1 <= M <= 100
1 <= W <= 100
1 <= K <= 100
```

Example Input File

```
3
3 3 3
A B C
D E F
G H I
ABCABCA
DHCD
BFAEIG
5 5 4
R A C E D
D O T F X
T S A E I
B Y D D P
L M N Y T
LATE
PATE
CARD
NEXT
7 7 6
C A T T L E M
O A M O E E R
A S A O M R R
D R U M T P M
C E R M E J E
R G H J K R M
S S R F T E I
AMERICA
CATTLE
STOMPER
RAREST
DRUM
DEAL
ROAR
```

Example Output to Screen

```
110
1011
1110101
```