University Interscholastic League

Computer Science Competition

2014 Regional 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 may be reworked and resubmitted, but will receive a deduction of 5 points for each incorrect submission. 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. Alphabet

Program Name: Alphabet.java Input File: alphabet.dat

Being a programmer and all, we have to know whether or not sequences of parentheses are properly ordered. For example, we know that "(())()" is properly ordered; however, ")()(" is not. But what if instead of parentheses, we have letters?

This is a bit different. If we have letters, then we have to know what characters match one another. Well, this is simple; it's just going to be the character on the opposite half of the alphabet. For example, A and Z match, B and Y match, C and X match, and so on, until you see that M and N match. Furthermore, they can match forward and backward. This means that "AZ" and "ZA" are properly ordered sequences of letters. And for a more complicated example, "ZNMA" would also match.

Input

The first line will contain the number of test cases to follow. Each test case contains a string of uppercase letters.

Output

Output "YES" if the string of letters is properly ordered; otherwise, print "NO".

Example Input File

```
4
AZ
ZA
CCXX
BMYN
```

Example Output to Screen

```
YES
YES
YES
NO
```

2. Average

Program Name: Average.java Input File: average.dat

This program will take a list of numbers and return the square root of the average of the squares of the numbers. This means you will take each number in a list, square it, and add it to a sum. Divide this sum by the number of elements in the list and take the square root of that number. Print this number to three decimal places on its own line.

Input

The first line will contain a single integer T. T lines will follow, each with a list of floating point numbers.

Output

Compute the answer for each line based on that list of numbers. Print that number on a separate line, rounding to three decimal places.

Example Input File

```
4
10.12 13.423 12.2134 33.1 44.2
2.2 3.3 4.4
3.547 6.816 8.453 3.218 7.159 9.032 3.118
0.500 0.910 0.737 0.585 0.891 0.660 0.819
```

Example Output to Screen

```
26.386
3.420
6.363
0.743
```

3. Convert Me

Program Name: ConvertMe.java Input File: convertme.dat

Given two positive whole numbers N and M. Find the minimal number of operations to convert N to M. The only operation allowed is:

- For any number N, any of its divisors apart from 1 and N can be added to it.

For example, if N = 4 and M = 8, 2 can be added to 4 twice to equal 8. For 10 and 25, 5 can be added 4 times

For example, if N = 8 and M = 36, 4 can be added 1 time to 8 to equal 12. Now, get factors of 12 which would be 2, 3, and 6. Using 6 would make N = 18. Add another 6 and N = 24. Then get factors of 24 one of which would be 12. 24 + 12 would make N = 36.

Input

There will be T test cases. Each test case will contain two integers N and M.

Constraints

```
N and M will be both <= 100, if N >= M, return -1  and  if M >= N, return -1
```

Output

Minimal number of operations to convert N to M using the given operation. If conversion is not possible, then output -1.

Example Input File

```
3
4 8
10 25
1 35
```

Example Output to Screen

```
2
3
-1
```

Explanation

In the first test case, 2 is the only divisor of 4 that is not 1 or 4, and it takes two steps using that factor to get to 8: 4 + 2 = 6, 6 + 2 = 8.

4. Descendants

Program Name: Descendants.java **Input File: descendants.dat**

You are working for a system administrator who finds that none of his tools are quite what he wants. He has a database with processes that have a name, process id, and a parent process id. There is a root process with an id of 1 and a parent id of 0. All processes will be descendants of this root process. You are going to print out a list of these processes in a nice format. You will first print root and then follow this format. Processes will have their children listed underneath them in alphabetical order. Processes will have a number of dashes before them equal to how far descended from the root they are.

Input

The first line will tell you how many processes will follow. Each process will be on its own line in the format of – [Name] [ID] [Parent ID].

Output

Output will consist of process names in the format specified above. Look at the example for a clear idea.

Example Input File

```
6
chrome 23 34
ps 2 1
sublime 4 34
awesome 34 1
eclipse 56 34
gdb 91 56
```

Example Output to Screen

```
root
-awesome
--chrome
--eclipse
---gdb
--sublime
-ps
```

5. Fantasy

Program Name: Fantasy.java Input File: fantasy.dat

You really like doing fantasy basketball, but since you don't want to use ESPN or Yahoo's sites, you decide to write your own. For the sake of clarity, we're going to call every player in the league a "participant". You decide to implement the mode of fantasy basketball called "head to head", where each week two participants get matched up, and either one of them wins, or it is a tie. Each participant has a team of N professional players, whose stats get added together throughout the week into a total for that participant.

These stats consist of 9 categories:

1. field goal percentage
2. free throw percentage
3. three pointers made
4. points
5. rebounds
6. assists
7. steals
8. blocks
9. turnovers.

For the first two stats, the higher percentage wins, for the next 6 the higher count wins, and for turnovers the lower number wins. Each participant gets a **fantasy point** for winning a category, and if a category is tied, neither participant gets points. Note: for field goals and free throws, if the number attempted is 0, they have a percentage of 0%. Also, the percentages are the team's percentage. For example, if one player is 6/6 and the other player on the team is 0/2, the participant's percentage is 6/8, or 75%. Your job is to write a program that, given these stats, determine which participant wins that week's match-up.

Input

- The first line will contain the number of test cases T.
- Each test case starts with one line, containing two strings and an integer N: The name of participant A, the name of participant B, and the number of players on each participant's team.
- The next N lines each contain one of participant A's players, and the next N lines each contain one of participant B's players.
- The format for a player is 11 space separated integers, in this order:
    - Field goals made
    - Field goals attempted
    - Free throws made
    - Free throws attempted
    - 3 pointers made
        - Points
    - Rebounds
    - Assists
    - Steals
    - Blocks
    - Turnovers.

Output

If one participant has more total **fantasy points** than the other, output "<Participant name> wins!", else output "Tie.".

Example Input File

```
2
Lebron Durant 1
15 22 2 5 1 33 7 3 4 0 8
10 22 7 10 1 28 8 3 2 0 5
Lebron Durant 2
12 20 9 9 1 34 3 3 0 0 3
16 23 6 8 4 42 9 6 2 0 3
12 23 5 5 4 33 7 5 2 0 3
15 30 10 12 2 42 3 10 2 0 5
```

Example Output to Screen

```
Tie.
Lebron wins!
```

Explanation of the example

Lebron's team has a field goal percentage of 15/22 = 0.68, while Durant's team has a field goal percentage of 10/22 = 0.45, so Lebron wins this category. Lebron has a free throw percentage of 2/5 = 0.4 and Durant has a free throw percentage of 7/10 = 0.7, so Durant wins this category. Durant and Lebron have the same number of three pointers made, so neither of them get a point for that. Lebron has 33 points whereas Durant has 28, so Lebron wins there. Lebron has 7 assists, whereas Durant has 8, so Durant wins that. They are tied in assists and blocks, but Lebron wins in steals whereas Durant wins in turnovers. This brings us to Lebron having 3 vs Durant having 3, with 3 ties, so it is a tie.

In the second test case, the totals become:
Lebron's team
28 43 15 17 5 76 12 9 2 0 6
Durant's team
27 53 15 17 6 75 10 15 4 0 8
Lebron wins FG%, they tie FT%, Durant has more 3PM, Lebron has more points and rebounds, Durant has more assists and steals, they tie in blocks, and Lebron has fewer turnovers. The total team points are thus 4 Lebron, 3 Durant, and 2 ties, so Lebron wins.

6. Hash

Program Name: Hash.java Input File: hash.dat

Hash tables are great for making a lookup table with "amortized constant time complexity lookup". It's okay if those words make no sense; they aren't important. Hash tables rely on hashing algorithms that assign unique indices to their input keys. The problem is that sometimes the algorithm will assign the same index for two different input keys. When this happens, a collision occurs. To solve a collision, the hash table might put one of the keys in an index nearby, but this does not always work either. You will use the following hash algorithm:

Given a String of characters [C0][C1][C2]…[C(N-1)], the index of this string (key) is calculated by summing each character's value multiplied by 2 to the power of its index in the String, and then taking a modulo the size of the table M. Each character's value is its distance from 'a'. In equation form,

$$index = \left(\sum_{i=0}^{N-1} (Ci - 'a') * 2^i \right) \% M$$

If this index is already taken, the key can be placed in the location directly above it, or directly below it (Pretend that the beginning and end of the table are connected). If both of these are taken, the collision cannot be resolved. Keys cannot be rearranged after being put in and must be added in the order given. When a collision occurs, consider both if the key is placed either one above or one below. You will test both placements for any collision and return the minimum number of unresolved collisions that will occur in the best case scenario.

For example, in a size 3 hash table, with three elements, 'a', 'b', and 'c', the 'a' hashes to $(0*2^0)\%3$, which equals 0, so the 'a' goes into position 0 in the hash table. Likewise, 'b' and 'c' hash to 1 and 2 and go into those table positions, with no unresolved collisions.

In another example, with 4 elements and a size 5 table, the element "afd" would hash to the following:

$$((0*2^0) + (5*2^1) + (3*2^2))\%5 = (0 + 10 + 12)\%5 = 22\%5 = 2,$$

placing it into position 2 of the table. An element "ba" would hash to position 1, and "ccs" would go into position 3. A fourth element "eee" would hash to the same location containing "ccs", position 3. But since the position just above (location 4) is available, "eee" goes there, and all is well with the world.

However, if the fourth element was "dee", it would hash to the value 2, which already contains "afd". To resolve this collision, you would look directly above and below, but those two positions are also taken, and therefore "dee" has nowhere to go, and we have 1 unresolved collision.

Input

The first line will contain T, the number of unique test cases. Each test case will start with two integers M and N, where M is the size of the hash table, and N is the number of strings to put into the hash table. This is followed by N lines, each with an input key string.

Constraints

```
0 < T < 30
0 < M < 20
0 < N < 20
M >= N
```

Output

For each test case, print the number of unresolved collisions that will occur in the best case.

Example Input File

```
3
3 3
a
b
c
5 4
afd
ba
ccs
eee
5 4
afd
ba
ccs
dee
```

Example Output to Screen

```
0
0
1
```

7. Monte Carlo

Program Name: MC.java Input File: mc.dat

In computational chemistry and physics, Monte Carlo simulations are very popular. For example to derive certain properties, particles might be laid out in some random distribution, and then random movements will be considered. A particle will be moved if it does not overlap with any other particles after the attempted movement. Otherwise the movement will not occur.

You will be given a list of particle locations and attempted movements. If a movement is successful, you will move the particle, otherwise you will not. Each position is the center of a circular particle, and all particles have a diameter of length one. Movements must be tested in the order that they are given.

Input

The first line will contain two integers P (the number of particles) and M (the number of attempted movements). P lines will follow. Each line will contain two floating point numbers, X and Y, giving the initial positions of each particle. M lines will follow after that, each line containing one integer and two floating point numbers. The integer, k, is the particle to move, where 1 is the first particle whose coordinates were given. The other two numbers are dx and dy (delta x and delta y), representing the changes in the x and y coordinates respectively.

Constraints

```
0 < P < 100
0 < M < 100
-100 < X < 100
-100 < Y < 100
0 < k <= P
-10 < dx < 10
-10 < dy < 10
```

Output

For each test case, print the location of each particle, in the order they were first given, to 3 decimal places in the form "X Y".

Example Input File

```
3 3
1.4 2.3
0.8 3.2
-1.0 21.9
2 0.3 -1.0
1 1.2 2.3
3 -9.8 4.2
```

Example Output to Screen

```
2.600 4.600
0.800 3.200
-10.800 26.100
```

Explanation

The first movement (particle 2) caused an overlap, so it did not occur. The next two movements (particles 1 and 3) did not cause any overlap, so they occurred.

8. Probe

Program Name: Probe.java Input File: probe.dat

Imagine that in the year 2185, you are exploring the galaxy looking for dilithium crystals. You have a scanner that can tell you at a given sector on the planet, how many dilithium crystals are in that area. You also have two probes that can each go down to the planet and mine the crystals in a 3x3 sector area. Once a probe mines a sector, that area is depleted. Since you only have two probes, you want to maximize the amount of crystals you can get. Given the scanner data for the planet write a program that tells you the maximum number of dilithium crystals you can get with your two probes.

Input

The first line will contain the number of test cases T.
The first line of each data set is a single integer N, the side length of the square sector of the planet area you wish to scan. The next N lines each contain N space-separated integers, which are the scanner readings for that sector, indicating how many dilithium crystals are in each part of the sector.

Output

For each test case, output the maximum number of crystals you can obtain with your probes.

Constraints

```
1 <= T <= 10
3 <= N <= 20
1 <= crystals in each sector <= 10^5
```

Example Input File

```
2
5
1 1 1 1 1
1 5 1 1 1
1 1 1 1 1
1 1 1 2 1
1 1 1 1 2
6
9 9 9 1 1 1
9 9 9 1 1 1
9 9 9 8 8 1
1 1 8 8 8 1
1 1 8 8 8 1
1 1 1 1 1 1
```

Example Output to Screen

```
23
145
```

Explanation of the example

If you send your first probe to (1, 1), it mines 13 crystals from there, then your planet looks like:

```
0 0 0 1 1
0 0 0 1 1
0 0 0 1 1
1 1 1 2 1
1 1 1 1 2
```

Then if you send your last probe to (3,3), and it mines 10 crystals from there, leaving the planet as

```
0 0 0 1 1
0 0 0 1 1
0 0 0 0 0
1 1 0 0 0
1 1 0 0 0
```

And you have mined a total of 23 crystals. There are no other two-probe positions that give you more crystals, so the answer is 23. For example, (1, 2) and (3, 3) give you 22, and (1, 1) and (3, 2) gives you 21.

9. Shirts

Program Name: Shirts.java Input File: shirts.dat

Since you are a fashionable computer scientist, with a measure of OCD in your personality, you decide to keep your shirts in your closet in a very specific order. First you sort your shirts by color, with this order from left to right: red ("R"), orange ("O"), yellow ("Y"), green ("G"), and blue ("B"). Among shirts that are the same color, you sort them by size, smallest to largest ("S" < "M" < "L"). Also, since you're a computer scientist, each of your shirts had a unique ID, so if they are the same color and size, the one with the smallest ID goes first. You just did your laundry, so rather than sort them yourself you'll write a program to do it. Write a program that, given the shirts in your laundry, outputs the ID's of the shirts in sorted order.

Input

The first line will contain the number of test cases T.
Each test case will begin with a single line, the number of shirts in your laundry pile, N.
Each of the next N lines represents a shirt in the following format: ID, color, size.

Output

For each test case, sort the shirts in the specified order, and then output the sorted shirt IDs, separated by spaces, on a single line.

Constraints

```
1 <= T <= 10
0 <= N <= 20
```

Example Input File

```
2
2
1 R M
2 O S
5
8 G M
12 G S
5 R M
15 Y L
6 Y L
```

Example Output to Screen

```
1 2
5 6 15 12 8
```

Explanation of the example

In the first test case, R comes before O, so shirt 1 comes before shirt 2.
In the second test case, R comes first, so shirt 5 is first. Since Y comes before G, 6 and 15 are next. 6 and 15 have the same color and size, so we order them by ID, so the next two are 6 and 15. Finally, we have the two green shirts 8 and 12. 12 is an S and 8 is an M, so 12 comes before 8. Thus, the final order is 5 6 15 12 8.

10. Volcano

Program Name: Volcano.java Input File: volcano.dat

You've been living on a remote rectangular island for the past 5 years now, and it has always worried you that there's a volcano nearby. You want to determine if you need to move your home to a different island, or if you are safe where you are.

A 2D grid filled with grass, trees, water, and a volcano represents your island. A T within this grid represents a tree, a W represents water, and the V represents the volcano. Your home is located at an S. Assume that in a single unit of time, you can move up, down, left, or right (but not diagonally) within the grid. However, you cannot jump in the water, or climb a tree. At the same time that you move, the volcano's lava spreads in all directions (up, down, left, and right, but not diagonally) from where it currently is, and lava moves at the same rate as you do -- one unit of distance in one unit of time. Lava may pass unhindered through trees, but cannot get past water. If your home can be reached by lava, then you must make it to the cave, C before the lava gets to you.

Determine if you will be able to reach safety if the volcano erupts one day, or if you need to move somewhere else.

Input

The first line of input contains an integer T, which represents the number of test cases. For each test case, there is a line containing two integers N and M, which represent the number of rows and columns in your 2D grid island. The next N lines each contains an M length string consisting of these characters:

T = Tree W = Water S = Home C = Cave V = Volcano . = Grass

Output

Output YES if you need to move your home to a better location, meaning that you can't make it to the cave safely should there be an eruption. Otherwise, if your home keeps you safe in its current location, output NO.

Example Input File

```
3
5 5
WWS.T
..T.C
.....
.W...
.VWW.
3 10
WWWCTTT...
WWW.TTT...
S...TTV...
1 6
VWC.TS
```

Example Output to Screen

```
NO
YES
NO
```

11. Wall-D

Program Name: Walld.java Input File: walld.dat

Wall-E's ancestor Wall-D was used when there was less trash. Wall-D would always try to get a specific amount of trash. You will help Wall-D find the length of the shortest path to obtain a certain amount of trash.
You will be given a square matrix (0 indexed) of size n x n. Wall-D starts at (n/2, n/2) (both floored). Wall-D must also pick up all trash he is ever on top of. He can move up, down, left, or right to find k trash. Each cell in the matrix is the amount of trash in that spot. Cells can be used multiple times and will have the same value each time they are used.

Input

The first line will contain the number of test cases T.
Each test case starts with a line containing two numbers. The first number is a side-length of the matrix n, and the next number is the amount of trash Wall-D wants to pick-up.
The next n lines show the matrix, with each cell separated by a space.

Output

Output the length of the shortest path (where 0 means that Wall-D starts on the desired amount of trash), or -1 if it is not possible.

Constraints

```
1 <= T <= 20
0 <= n <= 20
1 <= k <= 10^9
```

Each cell is a positive integer.

Example Input File

```
3
3 8
1 3 4
2 5 6
1 2 4
3 9
2 4 6
4 6 2
6 2 4
4 5
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
```

Example Output to Screen

```
1
-1
4
```

Explanation of the example

Start at 5 and move to 3; this sums to 8 and is a path length of 1. 5-2-1 also sums to 8, but is a longer path.
For the second case, the desired sum is odd, but all trash amounts are even, so it is impossible.

12. X, O and R

Program Name: XOR.java Input File: xor.dat

Mr. O and Mr. R approach Mr. X for a job in his jewelry store. After looking them over, Mr. X decides to test them. X has many diamond earrings in stock, each of different stone weights. Each earring is separately tagged with the price of the pair. For example, if there is a pair that costs $100, each of the two is tagged 100. There can be many pairs that cost the same.

Mr. X lays out all his earrings in a linear array after mixing them up, but craftily removes one earring. He then asks O and R to tell him the price of the pair that is missing one. Mr. O and R are not very smart, and ask you. Please help the poor guys!

Input

The first line contains T (the number of test cases). Each test contains an odd positive integer `n` denoting the size of the array, followed by `n` space separated integers denoting prices.

Output

Output the price of the incomplete pair.

Example Input File

```
4
3
2 2 1
9
2 3 3 4 3 4 3 2 2
5
2 3 5 2 5
5
9 9 6 6 8
```

Example Output to Screen

```
1
2
3
8
```