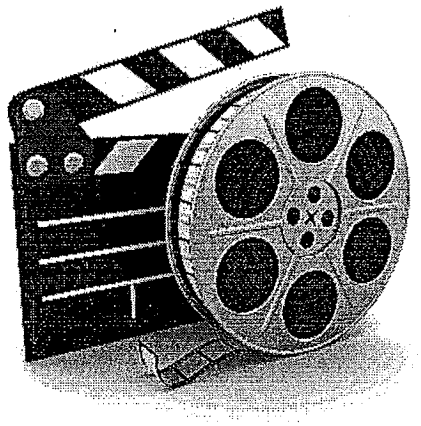
Problem 1: Movie Marathons

Source file: `movies.(c|cpp|java)`
Input file: `movies.in`

Suppose you have the following collection of movies:

- Movies A, B, and C are each 60 minutes long
- Movie D is 120 minutes long



You want to create a movie marathon from your collection. We'll define a marathon as a selection of movies that add up to a certain desired total runtime, without regard to order. Each movie can be included only once in a marathon, but multiple movies with the same runtime can be included. In this case, there are four two-hour marathons possible:

1. A, B
2. A, C
3. B, C
4. D

Given a collection of movies, you want to determine how many marathons of a specific runtime exist.

Input File

The first line of the input file consists of an integer N ($1 \leq N \leq 100$) indicating the number of test cases.

Each test case consists of two lines. The first line contains two integers, separated by a space: M ($1 \leq M \leq 20$) is the number of movies in the collection, and R ($1 \leq R \leq 1440$) is the desired total runtime in minutes of the marathon. The second line contains M space-separated integers in the range [1, 240] indicating the runtimes in minutes of each movie in the collection.

Output

For each test case, output a single line with the number of possible ways to create a marathon from the given data.

Sample Input File

```
2
4 120
60 60 60 120
4 200
60 60 60 120
```

Sample Output

```
4
0
```

Problem 2: Highly Composite Numbers

Source file: `composite.(c|cpp|java)`
Input file: `composite.in`

From Wikipedia, a *highly composite number* is "a positive integer with more divisors than any smaller positive integer has." For example, 4 is highly composite since it has 3 divisors (1, 2, 4). This is more divisors than 3 (1, 3), 2 (1, 2), and 1 (1). The smallest highly composite number is 1.

Your task is to determine whether a number is highly composite. If not, also determine the next lowest and next highest numbers that are highly composite.

Input File

The first line of the input file consists of an integer N ($1 \leq N \leq 100$) indicating the number of test cases.

Each test case consists of an integer D ($1 \leq D \leq 700{,}000$) on its own line.

Output

For each test case, output a single line. This line should contain the integer D by itself if it is highly composite. If D is not highly composite, the line should have the next lowest and next highest numbers that are highly composite, separated by a space.

Sample Input File

```
4
1
2
3
4
```

Sample Output

```
1
2
2 4
4
```

Problem 3: A World Devoid of Color

Source file: `colors.(c|cpp|java)`
Input file: `colors.in`

(I realized too late that on these printouts, the color image on the left is probably going to look very similar to the grayscale image on the right. Oh well!)

Digital images are commonly stored as *pixels*, each of which is assigned a certain color. Colors can be represented as some combination of red, green, and blue (RGB). If each component is allowed 8 bits of data, the possible values range from 0 to 255. For example, a red pixel would be stored as the ordered triple (255, 0, 0), while a blue pixel would be stored as (0, 0, 255). When the RGB components are all equal, the result is a shade of gray with (0, 0, 0) being black and (255, 255, 255) being white.

To remove the color from an image, we can convert it to grayscale. Many algorithms exist for this. One simple method is to replace each pixel's RGB components with the average of its max and min components. For example, a pixel that is originally (200, 40, 190) would be replaced by (200 + 40) / 2 = 120 on all three components.

Input File

The first line of the input file consists of an integer N ($1 \leq N \leq 100$) indicating the number of test cases.

Each test case begins with one line containing two integers R ($1 \leq R \leq 100$) and C ($1 \leq C \leq 100$), indicating the vertical and horizontal dimensions, respectively, of the image. Each of the following R lines contains C ordered triples in the format (r,g,b) where each of r, g, b is an integer in the range [0, 255]. Each triple (including the last one on each line) is followed by one space. Note that the integers inside each triple are separated by only commas, with no spaces.

Output

For each test case, output R lines of C integers each containing the grayscale values of the image pixels, using the average method above. Integer division should be used. The C integers in each line should be separated by spaces. Include one space at the end of each line.

Sample Input File

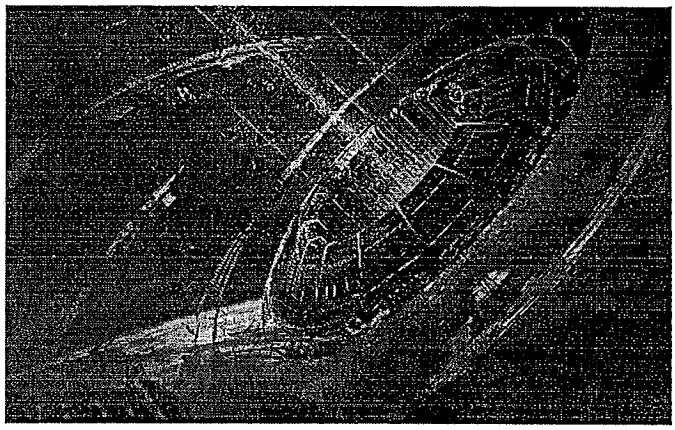
```
1
2 2
(200,40,190) (82,101,111)
(0,1,80) (182,33,90)
```

Sample Output

```
120 96
40 107
```

Problem 4: Galactic Conundrum

Source file: `galactic.(c|cpp|java)`
Input file: `galactic.in`



The year is 2317, and you've just been hired as Chief Operations Engineer for the Federation of Spacefaring Civilizations. Your first task is to oversee construction of new warp gates that will enable fast travel between different star systems in the galaxy. Due to space politics, it's more difficult to connect certain systems than others. For example, the Rigellians have had a longstanding feud with the Arcturians, so building a gate between those two systems is expensive and risky. You've been provided with data indicating the cost of constructing a gate between numerous pairs of systems.

Your task is to determine the minimum possible cost for constructing a set of warp gates that allow travel from any system to any other system. This travel need not be direct; for example, a route from Rigel to Arcturus could include one or more intermediate systems in between.

Input File

The first line of the input file consists of an integer N ($1 \leq N \leq 100$) indicating the number of test cases.

Each test case begins with an integer D ($1 \leq D \leq 100$) indicating the number of data points available. The next D lines contain information on the cost of connecting pairs of systems. Each of these D lines contains three parts, separated by spaces: the name of the first system, the name of the second system, and an integer C ($1 \leq C \leq 999$) indicating the cost of building a warp gate between those two systems. The names of each system will never contain spaces, and any pair of systems will appear at most once.

Output

For each test case, output a single line with the minimum possible cost to construct warp gates such that travel from any system to any other system is possible. If this is impossible with the given data, output INFINITY.

Sample Input File

```
2
3
Arcturus Rigel 400
Sirius Arcturus 6
Rigel Sirius 2
3
Arcturus Rigel 400
Sirius Arcturus 6
Betelgeuse Sol 10
```

Sample Output

```
8
INFINITY
```

Problem 5: World Traveler, Round 2

Source file: `world.(c|cpp|java)`
Input file: `world.in`

Ellie, our intrepid world traveler from last year's problem set, is back! She wants to visit several cities around the world, spending a certain number of days in each city. For example, suppose she wants to stay in Little Rock for two days and Batesville for one day. She could take any of these itineraries:

1. Little Rock (1 day) → Batesville (1 day) → Little Rock (1 day)
2. Little Rock (2 days) → Batesville (1 day)
3. Batesville (1 day) → Little Rock (2 days)

Given a list of Ellie's target cities and the desired lengths of her stays, you want to determine the number of possible itineraries.

Input File

The first line of the input file consists of an integer N ($1 \leq N \leq 100$) indicating the number of test cases.

Each test case starts with a single line containing an integer C ($1 \leq C \leq 20$), indicating the number of cities to be visited. The following C lines begin with the name of the city. City names may or may not contain spaces. Each city name may also be followed by a space and an integer D ($2 \leq D \leq 7$), indicating the number of days to stay in that city. If a city name is not followed by an integer, Ellie wants to stay there only one day.

Output

For each test case, output a single line indicating the number of possible itineraries.

Sample Input File

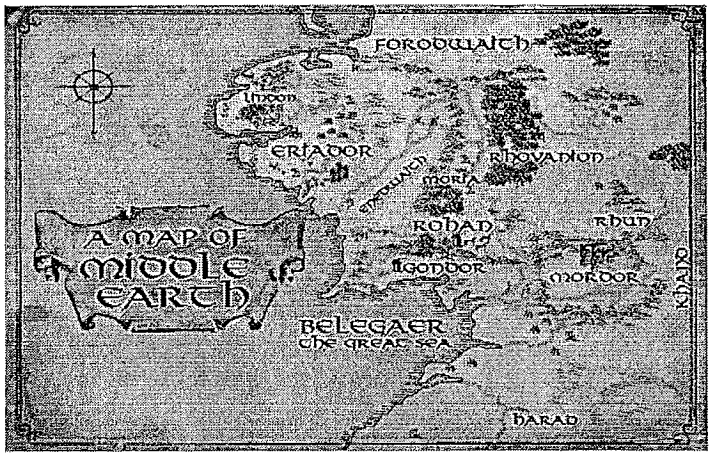
```
2
2
Little Rock 2
Batesville
3
Memphis
Little Rock
Batesville
```

Sample Output

```
3
6
```

Problem 6: Journey across Mordor

Source file: `mordor.(c|cpp|java)`
Input file: `mordor.in`



Frodo and Sam are traveling across Mordor on an epic quest to destroy the One Ring. They are running perilously low on supplies and energy, so they're looking to conserve as much strength as possible during their final push. Since clambering up steep cliffs and down deep ravines is difficult, they want to find a path that minimizes their cumulative elevation change.

We will model Mordor as a rectangular grid of locations, each of which has a certain elevation. Frodo and Sam will start at any row in the westernmost column. Their objective is to make it to any row in the easternmost column. Suppose they are currently at row r, column c. To advance to column $c + 1$, they must decide whether to maintain row r (head due east on the grid), go to row $r - 1$ (head northeast on the grid), or go to row $r + 1$ (head southeast on the grid). They do not want to go in any other direction, for fear of roving bands of orcs, spiders, and other unscrupulous creatures.

Your task is to find the smallest possible cumulative elevation change from the western edge to the eastern edge of the terrain. Both increases and decreases in elevation should be counted equally towards the cumulative change – for example, if they climb 50 feet in one step and drop 150 feet in the next, the cumulative elevation change is 200 feet.

Input File

The first line of the input file consists of an integer N ($1 \leq N \leq 100$) indicating the number of test cases.

Each test case starts with a single line containing two space-separated integers: R ($1 \leq R \leq 1000$) is the number of rows in the terrain, and C ($1 \leq C \leq 100$) is the number of columns. The next R lines each contain C space-separated integers, indicating the elevations of those points on the grid. These elevations are in the range [-9999, 9999].

Output

For each test case, output a single line indicating the minimum possible cumulative elevation change across that terrain.

Sample Input File

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

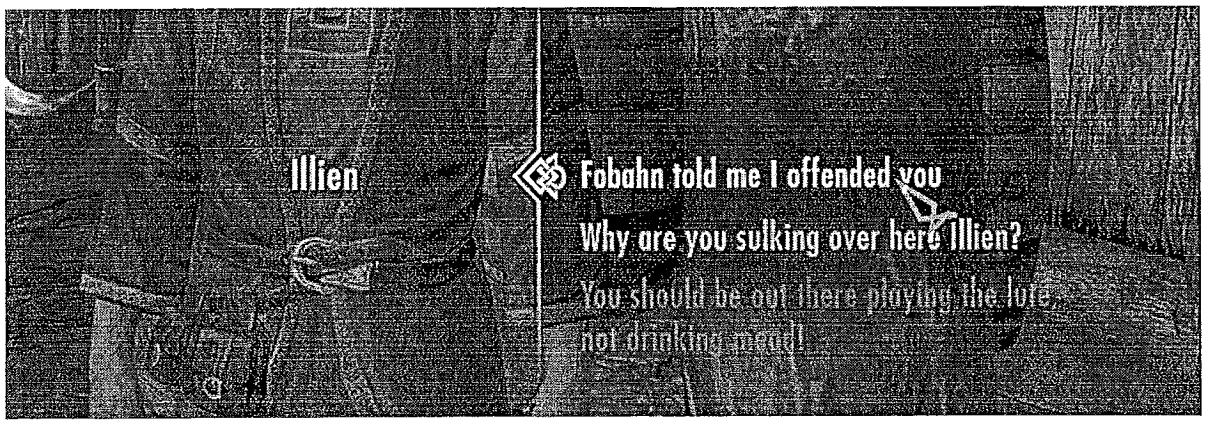
Sample Output

```
20
4
```

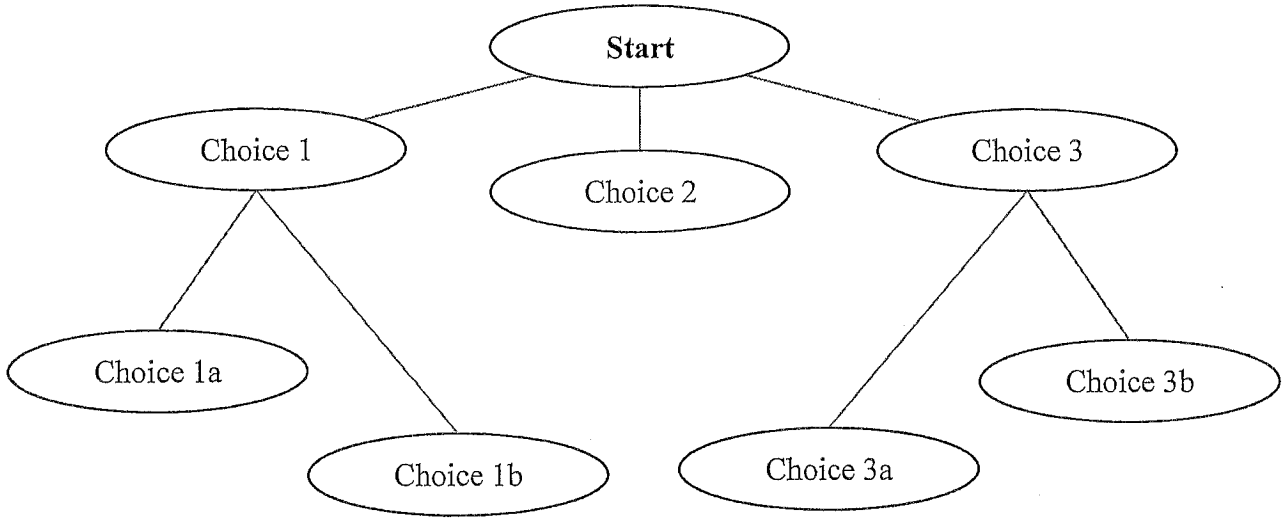
Problem 7: So Much to Say

Source file: `dialog.(c|cpp|java)`
Input file: `dialog.in`

You are writing a role-playing game where the player speaks to characters in the world via dialog choices. Each choice unlocks different dialog options. If you've played any games in the Fallout or Elder Scrolls series, this is similar to how conversation works in those titles.



This dialog system can be represented using a tree, as shown below. Initially the player can select Choice 1, Choice 2, or Choice 3. If s/he selects Choice 1, s/he can then select Choice 1a or Choice 1b, and so on. We'll assume that the player can never go back up the tree, only down.



You want to determine whether your dialog options are well balanced. This will be done through three metrics: the length of the shortest possible path through the dialog tree, the length of the longest possible path through the dialog tree, and whether the dialog tree is symmetric. A *symmetric* tree is structurally identical when mirrored about the root.

In the tree above, the shortest possible path has a length of 1 (Choice 2). The longest possible path has a length of 2; there are four such paths (Choice 1 → Choice 1a, Choice 1 → Choice 1b, Choice 3 → Choice 3a, Choice 3 → Choice 3b). The tree above is symmetric. However, it would <u>not</u> be symmetric if any one of Choices 1a, 1b, 3a, or 3b were removed.

Input File

The first line of the input file consists of an integer N ($1 \leq N \leq 100$) indicating the number of test cases.

Each test case begins with a single line saying **start**. Each line after that is a string representing one node in the dialog tree. Each string of dialog may contain any number of characters, including spaces. If the node has children, they are listed on the following lines, indented by one additional space. Hence, spaces at the beginning of the line indicate levels in the tree, with each space corresponding to one level. The test case ends with a single line saying **end**.

Output

For each test case, output a single line with three space-separated elements: the length of the shortest path through the dialog tree, the length of the longest path through the dialog tree, and whether the tree is symmetric (YES or NO).

Sample Input File

```
2
start
 Choice 1
  Choice 1a
  Choice 1b
 Choice 2
 Choice 3
  Choice 3a
  Choice 3b
end
start
 Choice 1
  Choice 1a
  Choice 1b
 Choice 2
 Choice 3
  Choice 3a
end
```

Sample Output

```
1 2 YES
1 2 NO
```