A+ Computer Science
Computer Science Competition
Hands-On Programming Set

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.

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.

5. A penalty of 5 points will be assessed each time that an incorrect solution is submitted. This penalty will only be assessed if a solution is ultimately judged as correct.

II. Point Values and Names of Problems

Number	Name
Problem 1	Advertising
Problem 2	Basalt
Problem 3	Boring
Problem 4	Digital Zoom
Problem 5	Diplomacy
Problem 6	Launch Date
Problem 7	Martian Potatoes
Problem 8	Phobos
Problem 9	Recruiting
Problem 10	Rocket
Problem 11	Stairs
Problem 12	Topography

1. Advertising

Program Name: Advertising.java **Input File: none**

A new relocation company is seeking clients to be one of the first to relocate to the Red Planet.

To make the billboard appealing, the marketing team wants to display the word MARS in large text and color each letter a different shade of red. They decide on maroon for M, auburn for A, bright red for R, and scarlet for S. You were then hired to design the prototype for their new billboard.

Output

Output the ad banner.

Example Output to Screen

```
+------------------------------+
|  M   M   AAA    RRR    SSSS  |
|  MM MM  A   A  R   R  S      |
|  M M M  A A A  RRRRR   SSS   |
|  M   M  A   A  R  R       S  |
|  M   M  A   A  R   R  SSSS   |
+------------------------------+
```

2. Basalt

Program Name: Basalt.java **Input File: basalt.dat**

A group at the space agency has recently discovered that much of Mars' surface is composed of basalt. Since then, they have sent teams to collect more data about the surface of Mars. Their process is as follows: the astronauts record the weight of each Basalt rock they can find, then the statisticians use this data to find patterns, and finally the engineers use these patterns to plan the designs of future structures on the terrain. However, the statisticians have been slowly increasing the price of their services over the last few years. It has gotten to the point that the space agency is no longer sees the value they are providing for the high cost. The group realized that these tasks seem rudimentary enough to automate it and hired you to do it.

Input

The first line will contain a single integer `n` that indicates the number of data sets that follow. Each data set will consist of a single decimal value ranging from 0.001 to 100,000 and will consist of no more than 3 decimal points.

Output

Output the weight of the lightest rock and the weight of the heaviest rock.

Example Input File

```
5
72.475
4.665
921.07
10836.12
7803.567
```

Example Output to Screen

```
4.665
10836.12
```

3. Boring

Program Name: Boring.java **Input File: boring.dat**

An astronaut on Mars decides to start writing a blog. While he considers himself a somewhat decent writer, he has discovered that most of his audience stops reading halfway into each blog post. He eventually narrowed the cause down to two words that appear too frequently: Space and Boring. In an effort to keep things interesting, you have offered to help.

Input

The first line will contain a single integer `n` that indicates the number of data sets that follow. Each line will consist of at least 5 characters with at least one being an alphanumeric character.

Output

If the line is boring, output `true`. Otherwise, output `false`.

Example Input File

```
8
Space
Boring
Space is boring
Roadster on mars, because we can
The engineer's pace
Typing is bORING
Too many spaces
muskrat
```

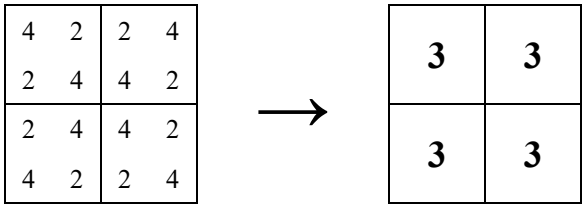
Example Output to Screen

```
true
true
true
false
false
true
false
false
```

4. Digital Zoom

Program Name: Zoom.java **Input File: zoom.dat**

The space agency has just landed a new rover on the Red Planet, but it had a few flaws. They recently discovered that their rover consistently keeps taking photos on the surface far too close to the subject. Normally, this can be fixed with an optical zoom lens. Unfortunately, this idea wasn't conceived during the initial design stages for some reason. They fired several executives as a direct result and decided to hire you to devise a solution. Since sending over the parts necessary for optical zoom to the planet and then assembling it there would be far too expensive, you decide to write a program by zooming in digitally.



To digitally zoom an image, a group of pixels are averaged in value. For this particular task, you will be averaging and rounding the values in each 2x2 area.

Input

The first line will contain a single integer `n` indicating the number of data sets that follow. Each data set will start with a single integer `s` denoting the size of the grid. The size of the grid will always be an even number with no more than 2 digits. Each element will be an integer, representing a pixel in the image, also with no more than 2 digits.

Output

Print out the result of zoomed image. There should be a space in between each integer in the grid and only one empty line after the last data set. Ensure that there is only one single empty line at the end of the output.

Example Input File

```
3
4
4 2 2 4
2 4 4 2
2 4 4 2
4 2 2 4
6
1 1 3 4 5 6
1 1 6 5 4 3
2 2 12 10 0 8
2 2 12 10 8 19
10 9 5 5 4 5
8 0 4 5 4 4
2
1 7
3 4
```

Example Output to Screen

```
3 3
3 3

1 5 5
2 11 9
7 5 4

4
```

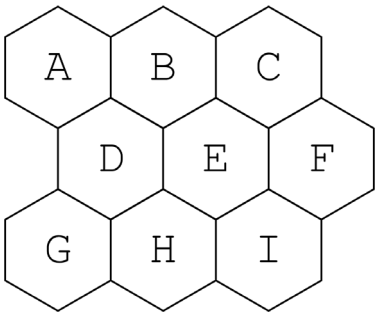
5. Diplomacy

Program Name: Diplomacy.java **Input File: diplomacy.dat**

Some world powers are looking to expand their territory beyond the lands of Earth. The space agency intends regulate who can claim territory claims on Mars before any conflict occurs. After nailing the interview, they decide to hire you, an unbiased and efficient diplomat, to resolve these matters.

This wouldn't normally be a problem if each country didn't mind who their neighbors were, but they do. Some countries have allies, or countries that they would prefer to be nearby. Some countries have rivals, or countries they would prefer to not share a border with. When assigning plots to countries, you should consider allies and rivals equally. The best arrangement is considered as the one with as many pairs of allies being neighbors and as little pairs of rivals being neighbors. Countries that have already been assigned plots cannot be moved.

In addition, plots are arranged in a hexagonal grid. This means that some plots may only have 2 neighbors while other plots may have up to 6 neighbors. To form a hexagonal grid from a list, you should start from the top left of the grid and fill each cell in that row before moving to the next row. For example, a 3x3 hexagonal grid formed by a list of the first 9 letters would look like the diagram below:



Input

The first line will contain a single integer `n` that indicates the number of data sets that follow. Each data set will consist with two integers `r` and `c` indicating the number of rows and columns in the hexagonal grid, respectively. There will always be at least 2 rows and columns, but no more than 8. The next line is a list of exactly enough countries to fill the grid. All country names will consist of only alphabetical characters with the exception of empty plots, which are marked by a `?` (question mark). The next line is a list of countries without an assigned plot yet. There will always be at least one country with an unassigned plot, but no more than half of the total countries. The last two lines of each test case represents a list of the allies and rivals, respectively. Allies and rivals are separated by a hyphen and each pair is separated by a space. There will always be at least one pair of allies and one pair of rivals.

Output

Output the grid with the best arrangement in the form of a list. All test cases will have a single solution with no ties.

Example Input and Output on the next page

Example Input File

```
3
3 3
? ? ? D E F G H I
A B C
A-B B-C
C-D A-F B-H
4 2
Green ? Red ? ? ? Orange Yellow
Purple Gray Blue Pink
Red-Purple Red-Blue Orange-Blue Orange-Gray
Green-Purple Blue-Yellow
3 3
I II ? III IV ? ? ? V
VI VII VIII IX
V-VII
II-VII II-VIII II-IX IV-IX III-VII
```

Example Output to Screen

```
A B C D E F G H I
Green Pink Red Purple Blue Gray Orange Yellow
I II VI III IV VII IX VIII V
```

6. Launch Date

Program Name: Launch.java **Input File: launch.dat**

The space agency was supposed to have launched a rocket into space towards Mars a month ago. However, a hurricane had passed through the city where the launch was and damaged much of the launch pad equipment. As repairs are finishing up, they decide to not take any chances for their next launch.

The agency indicates that days with 25% chance of rain or less is ideal for a launch day. In addition, they also prefer launch dates to fall on Fridays, Saturdays, or Sundays so they can get the maximum media coverage of the event. The earliest day that meets these two requirements should be chosen. If none of the available days meet those requirements, the earliest day with less than 25% chance of rain should be chosen. If there are still no available days, the day with the least chance of rain should be chosen.

Input

The first line will contain a single integer `n` that indicates the number of data sets that follow. Each data set will start with a single integer `s` denoting how many days are in each set. The next `s` lines will consist of the date and the chance of rain, separated by a hyphen. The list will always be in chronological order with the earliest day being first in the list and the latest day being last in the list

Output

Output the best launch day.

Example Input File

```
3
5
Wednesday, August 14 - 80% Rain
Thursday, August 15 - 50% Rain
Friday, August 16 - 30% Rain
Saturday, August 17 - 25% Rain
Sunday, August 18 - 0% Rain
4
Thursday, November 2 - 15% Rain
Friday, November 3 - 50% Rain
Monday, November 6 - 0% Rain
Thursday, November 9 - 0% Rain
4
Monday, January 20 - 20% Rain
Thursday, January 23 - 20% Rain
Monday, January 27 - 50% Rain
Thursday, February 5 - 5% Rain
```

Example Output to Screen

```
Launch Date: Saturday, August 17
Launch Date: Monday, November 6
Launch Date: Thursday, February 5
```

7. Martian Potatoes

Program Name: Potatoes.java **Input File: potatoes.dat**

Bruce is staying on Mars and wants to grow potatoes. The ideal temperature for growing potatoes on Mars is around 70˚F. While the temperature readers in the habitats read in Fahrenheit, only the main computer has access to change the temperature and it reads in Celsius. Bruce usually has to manually convert temperatures from Fahrenheit to Celsius and then subtract the current temperature from the ideal temperature each time he needs to change the temperature. After a few months, he started to get annoyed by the tedious process and decided to start complaining. To save these few steps, you decide to help Bruce by writing a program to simply indicate how much to change the temperature by.

The formula to convert Fahrenheit (represented as F) to Celsius (represented as C) is provided below:

$$C = (F - 32) \times 5 \div 9$$

Input

The first line will contain a single integer `n` that indicates the number of data sets that follow. Each data set will consist of a single integer `f` representing the value from temperature reader in the habitat. `f` will be no more than 100.00 and no less than 0.00 and will always be represented with 2 decimal places.

Output

Output how much Bruce needs to change the temperature by on the main computer and if it needs to be warmer or cooler than it is now. If the temperature is already at the ideal value, output `No change`.

Example Input File

```
5
68.14
82.29
70.00
67.97
70.01
```

Example Output to Screen

```
1.03 degrees warmer
6.83 degrees cooler
No change
1.13 degrees warmer
0.01 degrees cooler
```

8. Phobos

Program Name: Phobos.java **Input File: phobos.dat**

Scientists have received a strange signal apparently from Mars' largest moon. They have confirmed no astronauts have been there yet, making this signal even more mysterious.

Each message is dependent on whether or not Phobos is visible from the space agency's headquarters on Earth. You have deduced that when Phobos is visible from the time the message is sent, vowels always consist of at least 50% of the alphabetical characters from the message. If Phobos is visible, the consonants from the message are shifted one letter (skipping vowels) ahead and the vowels (skipping consonants) shifted one letter back. The shifts are opposite when Phobos is not visible. After the shifts, the message can be reversed to read the decrypted message.

Input

The first line will contain a single integer `n` that indicates the number of data sets that follow. Each data set will consist of only uppercase alphabetical characters and spaces.

Output

Output the decrypted message.

Example Input File

```
5
RMIOKE IQE IV
SUX VPUX MMUZ IF
RJ SOIZ AAT PUD AX
TSOI TE TICIJQ
EEEEG EG EG
```

Example Output to Screen

```
WE ARE ALIENS
DO YALL WANT WAR
WE CAN SEE YOUR HQ
PHOBOS IS OURS
HA HA HAAAA
```

9. Recruiting

Program Name: Recruiting.java **Input File: recruiting.dat**

The space agency is recruiting astronauts for a new Mars mission. They decide to automate part of the process since they expect a huge surge in the number of applications this year. For the recruitment process, they decide to emphasize on referrals from former astronauts from previous missions.

If an applicant has a first-degree connection with a former astronaut, they are awarded +1 point in the recruitment process. If they have a second-degree connection (a connection of one of their first-degree connections), they are awarded +½ points. If any connection was a former Commander, the points awarded are doubled. For example, if applicant A has a first-degree connection with a former astronaut and applicant B has a second-degree connection with a former Commander, both applicants are awarded +1 points. If another applicant C has a first-degree connection with a former astronaut, whom has a connection (second-degree for applicant C) with a former Commander, the applicant is awarded +2 points. In some cases, an applicant may be awarded points more than once from the same astronaut. If another applicant D has a first-degree connection with a former astronaut and a second-degree connection with the same astronaut, they are awarded +1½ points. All connections are mutual.

Input

The first line will contain a single integer `n` that indicates the number of data sets that follow. Each data set will consist of a single integer denoting how many lines that follow. This number will be at least 2 and less than 24. Each of the lines consists of the applicant/astronaut's name, followed by a list of first-degree connections they have. This list may be empty. Former astronauts will have the prefix `*` before their name and former Commanders will have the prefix `**` before their name.

Output

Output the applicants in descending order by the number of points they were awarded in the recruitment process, each separated by a space. If there is a tie between two applicants, use the applicant with the name that comes first alphabetically.

Example Input File

```
2
6
*01: 06
*02: 03
03: 02
04: 06
05: 04
**06: 01, 04
4
A1: A3
A2: A4, A3
A3: A2
*A4: A2
```

Example Output to Screen

```
04 03 05
A2 A3 A1
```

10. Rocket

Program Name: Rocket.java **Input File: rocket.dat**

While en route to the Red Planet, the rocket discovers a small asteroid field directly in their path. They were already a few days behind schedule due to technical failures on the launchpad, so they can't afford to take a longer route. Command control then decides to take some chances and send the rocket into the asteroid field.

The rocket must move 1 unit to the right each move to keep it on schedule. In addition to moving rightward, the rocket is can also move 1 unit vertically in either direction since the rocket is capable of quickly changing velocities. This allows the rocket to travel diagonally, as long as it is rightward.

Input

The first line will contain a single integer `n` that indicates the number of data sets that follow. Each data set will start integers `r` and c, indicating the rows and columns that follow. `#` denotes an asteroid and `.` (period) denotes a clear space for the rocket to travel through. The rocket needs to be able to travel from any point on the first column to any point on the last column.

Output

If the rocket is able to successfully traverse through the asteroid area to the intended destination, output `Destination: Mars`. Otherwise, output `Destination: Asteroid`.

Example Input File

```
3
5 5
..##.
#..##
#....
##..#
..#..
2 6
..#..#
...#..
4 4
#...
#.#.
#.#.
..#.
```

Example Output to Screen

```
Destination: Mars
Destination: Mars
Destination: Asteroid
```

11. Stairs

Program Name: Stairs.java **Input File: stairs.dat**

The Martian crew were traveling back from an outpost back to the habitat. They were just a few minutes away when they get their rover stuck near a steep slope. They try lifting one of the tires off of the ground, but it unexpectedly detaches from the rest of the rover and sends a crew member to slip down the slope. While his helmet was not damaged, his arms get stuck underneath a boulder. After the rest of the crew helps him out, they carry him to the habitat for examination. Once they get there, they realize that he would be unable to climb the ladder and needs a different method for him to get inside the habitat. Using resources nearby, they decide to create stairs to get him inside the habitat.

Input

The first line will contain a single integer `n` that indicates the number of data sets that follow. Each data set will start with an integer `x`, representing the size of the stairs, and two characters `d` and `m`, representing the direction the stairs are facing and the material the stairs needs to be made of, respectively.

Output

Output the stairs for the injured crew member to use. Ensure that there are no trailing spaces following the last character of each line.

Example Input File

```
3
3 R S
6 L M
8 R #
```

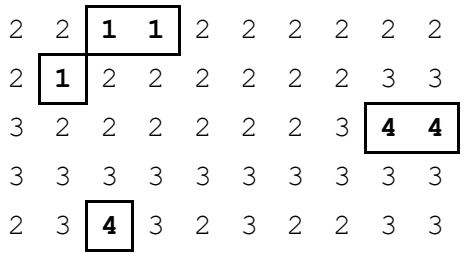
Example Output to Screen

```
  S
 SS
SSS
M
MM
MMM
MMMM
MMMMM
MMMMMM
        #
       ##
      ###
     ####
    #####
   ######
  #######
########
```

12. Topography

Program Name: Topography.java **Input File: topography.dat**

A group of cartographers are looking to map the topography of Mars to emphasize the extreme points of elevation areas on the terrain. In order for a group of coordinates to be considered an area, they must all have the same elevation level and they must all be horizontally or vertically contiguous (not diagonal). However, since these cartographers want to highlight the extreme points (highest and lowest points) of each map, they need someone to help and automate that task. They decided to hire you to write a program to find these relevant areas of the map.



In the example above, the highest elevation in the map has an elevation level of 4 and the lowest is 1. Four areas are highlighted to indicate the contiguous areas at the highest or lowest elevation.

Input

The first line will contain a single integer `n` that indicates the number of data sets that follow. Each data set will start with a two integers, `x` and `y`, denoting the number of columns and rows, respectively. Both `x` and `y` will be less than 25. The next `y` lines will each consist of `x` single digit integers, each representing the elevation level at that coordinate.

Output

For each relevant area, output the number of coordinates it consists of, the elevation, the coordinate closest to the top-left of the map, and the coordinate furthest from the top-left of the map. All coordinates in the output should be one-indexed. In addition, areas should be sorted by the shortest distance from its top-left coordinate to the top-left of the map. If there is a tie, the area closest to the top of the map takes precedence. Separate each test case by an empty line and ensure that there is only one single empty line at the end of the output.

Example Input File

```
3
10 5
2211222222
2122222233
3222222344
3333333333
2343232233
3 2
989
999
2 2
00
00
```

Example Output to Screen

```
2 coordinate(s) with elevation 1 from (3, 1) to (4, 1)
1 coordinate(s) with elevation 1 from (2, 2) to (2, 2)
1 coordinate(s) with elevation 4 from (3, 5) to (3, 5)
2 coordinate(s) with elevation 4 from (9, 3) to (10, 3)

5 coordinate(s) with elevation 9 from (1, 1) to (3, 2)
1 coordinate(s) with elevation 8 from (2, 1) to (2, 1)

4 coordinate(s) with elevation 0 from (1, 1) to (2, 2)
```