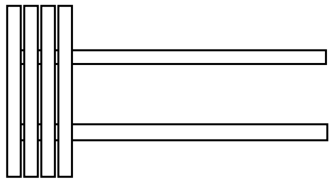
1. Backyard Fence

Program Name: Backyard.java   Input File: backyard.dat

The ABC Fence Company installs new fencing in customers' yards. Instead of installing pre-manufactured panels, the ABC Fence Company builds their fences one board at a time. Lucas, the job foreman, needs you to write a program for him so he will know how many boards and pickets he must to buy to install a particular fence.

For each fence, he needs two rails that run the length of the fence horizontally and are made from 6' lengths of 2" x 4" boards. To avoid waste, these boards may be cut and divided between the two rails if possible. For the purposes of this problem, assume that none of the length of the board is lost in a cut.



Additionally, he needs enough vertical boards, called pickets, to cover the length of the fence. The width of each picket is 1 3/4 inches and he leaves a 1/8 inch gap between boards to allow for expansion caused by weather. To allow for the possibility of damaged pickets, Lucas always buys some overage of one extra picket for every 10 feet, or part thereof, of fencing to be installed. Lucas does not order an extra picket to cover a partial picket width at the end of the fence since he can use his overage to cover that space. Since Lucas makes his own gates from the fencing materials, no extra boards or pickets are needed for gates.

You are to write a program to determine how many 2" x 4" boards he will need for the rails and how many pickets, including the overage, he will need to cover the rails.

Input

The first line of input will contain a single integer `n` that indicates the number of fences to be built. Each of the following `n` lines will contain a single integer that indicates the length in feet of a fence to be built.

Output

On a single line for each fence, print two integers in the form `b p` where `b` is the number of 6' boards he needs to buy and `p` is the number of pickets he needs to buy to build the fence.

Example Input File

```
4
125
275
93
154
```

Example Output to Screen

```
42 813
92 1788
31 605
52 1001
```

2. Burpees

Program Name: Burpees.java   Input File: burpees.dat

To enhance his agility, physical fitness, and aerobic fitness, Scott has joined the Burpee Challenge support group on Facebook. The members of this group have publicly challenged each other to do one Burpee on the first day of the challenge, two on the second day, three on the third day, and so forth. A Burpee is a full body exercise performed in four steps:

1. From a standing position, drop into a squat position with your hands on the ground.
2. Extend your feet back in one quick motion to assume a push-up position (doing a push-up is optional).
3. Return to the squat position in one quick motion.
4. Jump to an upright standing position.

Input
The first line of input will contain a single integer `n` that indicates the number of people in the challenge. Each of the following `n` lines will contain a single integer that indicates the number of days the person is going to participate in the Burpee challenge.

Output
For each person in the challenge, you will print the total number of Burpees that he will have done by the end of his challenge.

Example Input File
```
3
100
54
20
```

Example Output to Screen
```
5050
1485
210
```

3. Championship Swimming – UIL State

Program Name: Championship.java   Input File: championship.dat

In the UIL State championship swim meet, there are preliminary heats (or swims) conducted for an event during the day to determine the 16 fastest swimmers in the meet who will swim in the championship finals and consolation finals at night. The fastest eight swimmers from the preliminary heats swim in the championship finals race and the next fastest eight swimmers swim in the consolation finals race. The remaining swimmers do not swim in the night events.

In swimming, the lanes are numbered from 1 to 8 with lane 1 being on the far left and lane 8 being on the far right. The preferred lane numbers for a swimmer are in this order: 4, 5, 3, 6, 2, 7, 1, 8. By seeding the swimmers in this order, the fastest swimmers have the advantage of swimming in the smooth water in the middle of the pool and the slower swimmers have to fight the wake of the faster swimmers by swimming in the outside lanes.

You are to write a program that will find the 16 fastest swimmers from the list of preliminary swimmers and print their lane assignments for the consolation finals and championship finals races.

Input

The first line of input will contain a single integer `n` that indicates the number of events to be swum. For each event:

- The first line will contain the name of the event.
- The second line will contain a single integer `m` that indicates the number of swimmers in that event.
- Each of the following `m` lines will contain:
  - a swimmer's first and last name separated by a space but containing no other spaces, followed by a space,
  - the swimmer's time in the preliminary heat in the form: `mm:ss.hh` where `mm` is the number of minutes, `ss` is the number of seconds, and `hh` is the number of hundredths of a second, but
  - if the swimmer's time is `DQ`, the swimmer was disqualified from the event and cannot swim in the consolation finals or championship finals events.
- **Note**: You may assume that there are at least 10 swimmers that will qualify for the night events and no two swimmers have the exact same time.

Output

For each event:

- Seed the swimmers into their preferred lanes based on the lane order above.
- Then print the event and a space followed by `CONSOLATION FINALS`.
- On each of the next 8 lines (or less if there are not enough swimmers):
  - print the lane number, in order 1-8, followed by a period and a space,
  - print the last name, a comma and a space followed by the first name of the swimmer assigned to that lane, and then another space, and finally,
  - print the seed time of the swimmer as shown in the example on the next page.
  - Do not assign a lane to or print the name of any disqualified swimmers.
- Print a blank line.
- Then print the event and a space followed by `CHAMPIONSHIP FINALS`.
- On each of the next 8 lines (or less if there are not enough swimmers):
  - print the lane number, in order 1-8, followed by a period and a space,
  - print the last name, a comma and a space followed by the first name of the swimmer assigned to that lane, and then another space, and finally,
  - print the seed time of the swimmer as shown in the example on the next page.
- Print a blank line. A blank line is optional after the last data set.

(continued on next page)

3. Championship Swimming – UIL State (cont.)

Example Input File

```
1
100 M FREESTYLE
18
JAMES SMITH 01:10.01
TOM JONES 01:20.00
HOWARD COSELL 01:19.30
DON MEREDITH 01:21.28
RON JONES 01:17.30
RICHARD LUCAS 01:09.30
RONNIE WERTH 01:11.38
JACK MOSES 01:19.94
DAN ROGERS 01:15.84
ROGER WILLIAMS 01:15.83
WILLIAM GEORGE 01:12.27
GEORGE JONES 01:13.45
MARK CHRISTIE 01:45.12
CHRIS LEOPARD 01:59.29
MAJOR APPLEWHITE DQ
RICK CHARLES 01:12.67
CHARLES SHERMAN 01:13.22
SHERMAN WILLIS 01:17.78
```

Example Output to Screen

```
100 M FREESTYLE CONSOLATION FINALS
1 MEREDITH, DON 01:21.28
2 MOSES, JACK 01:19.94
3 WILLIS, SHERMAN 01:17.78
4 ROGERS, DAN 01:15.84
5 JONES, RON 01:17.30
6 COSELL, HOWARD 01:19.30
7 JONES, TOM 01:20.00
8 CHRISTIE, MARK 01:45.12

100 M FREESTYLE CHAMPIONSHIP FINALS
1 JONES, GEORGE 01:13.45
2 CHARLES, RICK 01:12.67
3 WERTH, RONNIE 01:11.38
4 LUCAS, RICHARD 01:09.30
5 SMITH, JAMES 01:10.01
6 GEORGE, WILLIAM 01:12.27
7 SHERMAN, CHARLES 01:13.22
8 WILLIAMS, ROGER 01:15.83
```

4. Cruise Ship

Program Name: Cruise.java   Input File: cruise.dat

To thrill his passengers, an Italian cruise ship captain wants to know how close he can get to a landmass without the cruise ship hitting the seabed, a sand bar, cliffs, or some other underwater obstacle. He has asked you to write a program to determine the closest his cruise ship can get to a particular landmass, given the ship's draft and an underwater seabed map of the region. The map will be represented by a rectangular matrix of cells.

The ship can move in any direction, including diagonally, but only above non-land cells that have a depth greater than the ship's draft (because if it's the same, you start scraping the ship along the bottom and your boss doesn't like that). The distance between the ship and the closest landmass is the number of cells between the two (including neither the ship or the landmass). Regardless of whether you are moving diagonally, horizontally or vertically across a cell, it counts as one cell.

Input

The first line of the input file will contain a single integer `n` that indicates the number of scenarios to follow. For each scenario:

- The first line will contain three integers `r c d` that indicate, in order, the number of rows in the map, the number of columns in the map, and the ship's draft. The ship's draft is the number of feet the ship floats below the surface of the water.
- The following `r` rows will each contain a combination of `c` number of integers or letters as described below, each separated by a single space:
  - An integer denotes the depth, in feet, of the water in that location.
  - The character `S` denotes the ship's current location (there will always be only one `S` in the map).
  - The character `L` denotes a landmass (there will at least be one `L` in the map)

Output

For each scenario you will print on a single line, the integer *X*, where *X* is the smallest number of cells between the ship and the closest landmass, followed by a space and the word `cell` or `cells` depending on whether *X* is one or not.

Example Input File

```
1
5 5 15
S 3 15 3 2
16 16 15 15 15
16 10 5 20 L
16 10 5 5 L
16 10 5 L L
```

Example Output to Screen

```
2 cells
```

5. Depth Sorted Rasterizer

Program Name: Depth.java   Input File: depth.dat

The company you work for needs a 3D rasterizer for a rendering program and you have been asked to develop it.

- A rasterizer is an algorithm that takes drawing commands and renders the 2D image they describe. Each draw command is comprised of a primitive (i.e., shape), the color used, and the primitive specific details, such as the location, width and height for a square, for example, or the starting and ending points for a line.
- Each primitive will have a depth floating point value that indicates how deep the object is into the scene. If you think about the image as being a picture being taken from a virtual camera, the depth value 0.0 is as close as you can get to the camera, and 1.0 is as far as the camera can see.
    - Anything less than 0.0 or greater than 1.0 cannot be seen by the camera.
    - An object at a depth of 0.5 would be obscured, either completely or partially, by an object at a depth of 0.3.
    - When a part of one object overlaps a part of another object and they have the same depth value then the primitive drawn last should be the one visible.
    - Your code should handle all these cases when generating the 2D image.

Before the code goes into the application your boss wants you to write a small program to test it. You will read information from a text file and output the image as text. Each line of the image description will contain a single primitive along with the necessary details to rasterize it. The primitive line formats you will need to support are described in the table below. For simplicity the colors in the image will be represented as uppercase characters A to Z.

Primitive Keyword and Details	Description
BOX color tlx tly brx bry z fill	color: the color character tlx: integer for the x coordinate of the top left corner tly: integer for the y coordinate of the top left corner brx: integer for the x coordinate of the bottom right corner bry: integer for the y coordinate of the bottom right corner z: the depth of the primitive fill: `Y` if you should fill the whole square with the color, `N` if you should just draw the 1 pixel outline
LINE color x1 y1 x2 y2 z	color: the color character x1: integer for the x coordinate of the starting point y1: integer for the y coordinate of the starting point x2: integer for the x coordinate of the ending point y2: integer for the y coordinate of the ending point z: the depth of the primitive **NOTES**: Either x1 and x2 will be equal, or y1 and y2 will be equal (no diagonal lines)
DIAMOND color cx cy s z	color: the color character cx: integer for the x coordinate of the center of the diamond cy: integer for the y coordinate of the center of the diamond s: the number of pixels the diamond extends in each direction from the center (so if it is 3, the diamond goes 3 to the left, 3 to the right, 3 up, and 3 down, making the distance of the longest points 7) z: the depth of the primitive **NOTES**: If the width or height is even, add the extra pixel to the right or the bottom (towards the maximum)

(continued on next page)

5. Depth Sorted Rasterizer (cont.)

Input

The first line will contain a single integer `n` that indicates the number of image descriptions to follow. Each image description will be composed of an unknown number of lines as follows:

- The first line contains 2 integers `w` and `h` indicating the width and height of the image.
- In rendering, screen resolution, position and size are done in width x height order, that is, the `x,y` coordinate of the top left of the image is always `0,0`, with `x` increasing positively to the right to `width-1` and `y` increasing downwards to `height-1`.
- Initially, the image should have all cells initialized to the period character '`.`' to denote that nothing has been drawn there.
- Read and process the commands one line at a time:
    - For the primitive keywords `BOX`, `LINE`, or `CROSS` found at the beginning of each input line, use the description from the table on the previous page corresponding to each keyword to interpret the details following the keyword on the input line. For instance, for the input line `BOX W 1 1 3 4 0.1 Y` the drawing program should draw a box from `1,1` to `3,4` filling the whole box with `W` at a depth of `0.1`.
    - Use the single word `END` to complete the current image by printing it.

Output

For each image you will output the image, a row per line, without any spaces between columns. There should be a blank line between images. Pixels in which nothing was drawn should be indicated by the period character (`.`).

Example Input File

```
1
10 8
BOX W 1 1 3 4 0.1 Y
BOX A 9 7 100 100 0.5 N
LINE L 2 4 8 4 0.6
DIAMOND X 5 4 3 0.7 N
END
```

Example Output to Screen

```
..........
.WWW.X....
.WWWX.X...
.WWW...X..
.WWWLLLLL.
...X...X..
....X.X...
.....X...A
```

6. Duck Arcade

Program Name: Duck.java   Input File: duck.dat

At the State Fair of Texas, there is a Duck Arcade game on the midway that guarantees that you will win a prize every time you play. In this arcade, there are hundreds of small, plastic ducks floating down a stream of water. On the bottom of each duck and unseen by the player is a positive integer in the range from 1 through 10. There is always an equal number of ducks with each of the 10 integers.

To play the game, the player selects 5 ducks out of the stream. The prize the player wins is based on the sum of the numbers on the bottom of the five ducks selected using the following guidelines:

- Level 1 – if the sum is 25 or less, the prize is a chalk dog
- Level 2 – if the sum is 26 through 30, the prize is a 4" stuffed animal
- Level 3 – if the sum is 31 through 35, the prize is a 12" stuffed animal
- Level 4 – if the sum is greater than 35, the player can choose any prize in the Duck Arcade.

Your boss wants you to write a program that simulates random games for a large number of players so he can determine if the sum of the numbers on the 5 ducks selected in each game are in a range that will allow him to make a healthy profit.

To simulate the random numbers on the ducks selected, you will construct an object of the type `java.util.Random`. This class allows you to specify the seed for the random number generator. For a given seed, the order of the random numbers is always the same.

Input

The first line of input will contain a single integer `n` that indicates the number of games in your simulation. The second line will contain a long integer that you will use to seed the random number generator for the simulation.

Output

You will print the number of games in the simulation that fall into each of the Levels, 1 through 4, following the format shown in the example output below.

Example Input File

```
25
43234564324455
```

Example Output to Screen

```
Level 1: 10
Level 2: 6
Level 3: 4
Level 4: 5
```

Note: these are the random numbers generated for the 25 games in the input file:

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

7. DVD Elevator 2012

Program Name: DVDElevator2012.java **Input File: dvdelevator2012.dat**

You have taken an internship at Honey Badger Games working on a launch title for the new Phony BoxStation video game console. The game is currently loading too slowly and the lead programmer has asked you to implement an elevator loading algorithm to speed it up.

Data on optical disks, such as DVDs, are stored in a linear fashion from the inside to the outside of the disk. The laser always reads data in multiples of sector sizes as the laser moves across the disk, also from inside to outside. The laser cannot read any data when it is moving from the outside to the inside although it does take time to move the laser. You don't have direct control over the laser; you can only send file read requests to the operating system, which will process the read requests in the order it receives them.

An elevator algorithm is used to order requests in order to minimize unnecessary laser motion and increase disk read throughput by reading data as much of the time as possible when the laser is moving. Your boss has given you explicit instructions on how to implement the algorithm:

- By default the algorithm should be in an "idle" state while waiting for read requests.
- There are several rules for the laser position:
  - You can assume the operating system starts by having the laser positioned at the beginning of the first sector.
  - When a laser reads it starts at the beginning of a sector, and at the end of the read is positioned at the end of the sector just read (which is the beginning of the next sector). That means if the laser ends a read on sector 100 it is now actually on sector 101, so a read for sector 100 would require moving backwards.
  - In order to optimize reads, the algorithm should keep track of what would be the position of the laser at the end of the last read request sent to the operating system, hereafter referred to as the LEP (Laser End Position). Since the operating system may be in the middle of a read, the LEP is not necessarily the same as the current laser position.
- When a read request is received there are two possibilities for processing that request:
  - If the read would position the laser to start reading at or after the LEP, so that the laser can read immediately or move forward to read the data, the read request should be sent off right away to the operating system where it will be added to the end of the other requests.
  - If the read would position the laser to start reading before the LEP, so that the laser would have to move backward to read the data, the read request should be added to a batch list, as described below.
- The batch list should be processed as follows:
  - If the batch list is empty when a request is to be added, a 10-millisecond timer should be started so that all other requests received during that time can be put into a single request batch (where 1 millisecond = 1000 microseconds).
  - If the batch list isn't empty the request should be added to the list in sector read order, so all the requested data can be read as fast as possible without additional seeks. You can assume that no reads in any given batch will have their sectors overlapping.
  - When the timer on a batch in progress expires all the requests in the batch list should be dispatched to the operating system.
  - **Note**: Any request received at the exact end time of the batch time window should either be sent off before the batch or become part of the batch, depending on the read sector of the request and the LEP.
- When there is no work to do the system returns to the "idle state" with the laser positioned at the LEP, as described above.

The lead has asked that you test your code by writing a program that encompasses your algorithm and reads fake data from a text file in the order requests would be given to your code during the game.

(continued on next page)

7. DVD Elevator 2012 (cont.)

Input
There are an unknown number of lines in the input file. Each line will contain 3 long integer values:

- the time in microseconds since the start of the game until the request was made,
- the start sector for the request, and
- the end sector for the request.

The microsecond start time for any given line is guaranteed to never be lower than any previous lines (i.e., you can assume the order of input lines matches a timeline that never moves backwards).

Output
You should output all the requests from the input file, one per line, in the order they are sent to the optical disk hardware. Each line should be of the form "`Time X: Y to Z`" where `X` is the time in microseconds since the start of the game until the request is sent and `Y` and `Z` are the start and end sectors for the request respectively.

Example Input File
```
10 2 200
100 300 400
900 240 270
1100 220 230
10900 410 430
10901 0 10
10902 12 13
10903 11 12
11000 450 500
11001 440 450
```

Example Output to Screen
```
Time 10: 2 to 200
Time 100: 300 to 400
Time 10900: 410 to 430
Time 10900: 220 to 230
Time 10900: 240 to 270
Time 11000: 450 to 500
Time 20901: 0 to 10
Time 20901: 11 to 12
Time 20901: 12 to 13
Time 20901: 440 to 450
```

8. Escape!

Program Name: Escape.java Input File: escape.dat

You are out in the woods, camping alone when you wake up to a loud roar. A hungry bear has come up onto your campsite and thinks you smell delicious! The woods are pretty thick and you want to know what chance you have of escaping alive. In order to do this you whip out your laptop and write a program that can take in various configurations of obstacles, the bear, and yourself and will compute your probability of escape. Your program will compute every permutation of moves you can make, a simulation of the bear's logic, and then calculate how many solutions result in your escape and how many result in your demise. You obviously program pretty quickly, as that bear sure is hungry!

You have designed your program so the woods are represented by rectangular grid composed of the following characters:

- `#` – an Obstacle, you cannot move into this space
- `E` – Escape space, you live!
- `B` – Bear!
- `Y` – You
- `.` – empty space, you can move here

Your program will work as follows:

- You and the bear will alternate moving. You will move first and every time you make a move, the bear will make one right after unless the chase is over.
- Both you and the bear can move in any direction (horizontally, vertically or diagonally), but you cannot go back to a square that you've been to before. The bear has no such restriction.
- The bear will always move toward you, as you smell like dinner.
- If your move is to the same row or column as the bear, he will move one square towards you, in that row or column.
- If your move is not to the same row or column as the bear, the bear will move diagonally toward you.
- The bear has no problem moving into the Escape space if that means it gets closer to its dinner.
- If your move is to a spot that is adjacent to the bear, then the bear's next move will be to move into your square to eat you. Yum!
- If the move the bear wants to make is into an obstacle, he will stay put and look frustrated.
- If you cannot move because all adjacent spaces have been visited or contain obstacles, then the chase is over and is considered neither an escape for you nor a dinner for the bear.
- Every grid will result in at least one successful escape for you or dinner for the bear.

Input

The first line will contain a single integer `n` that indicates the number of escape grids to follow. For each escape grid:

- the first line will contain 2 integers in the form `r c`, where `r` is the number of rows in the grid, `c` is the number of columns, and both `r` and `c` are greater than zero and less or equal to five.
- The next `r` lines will contain `c` characters as described above with no spaces.
- **Note:** Each grid is guaranteed to have only one Bear, only one You, and at least one Escape.

Output

For each grid you will output the truncated percent chance of survival (integer based), which is the percentage of the number of the solutions that ended in your escape divided by the number of solutions that were either your escape or you being dinner. Each percent should be followed by the % character and each grid's output should be on its own line.

(continued on next page)

8. Escape! (cont.)

Example Input File

```
2
5 5
##...
..Y..
.....
##...
BE...
4 4
...Y
..#.
..B.
...E
```

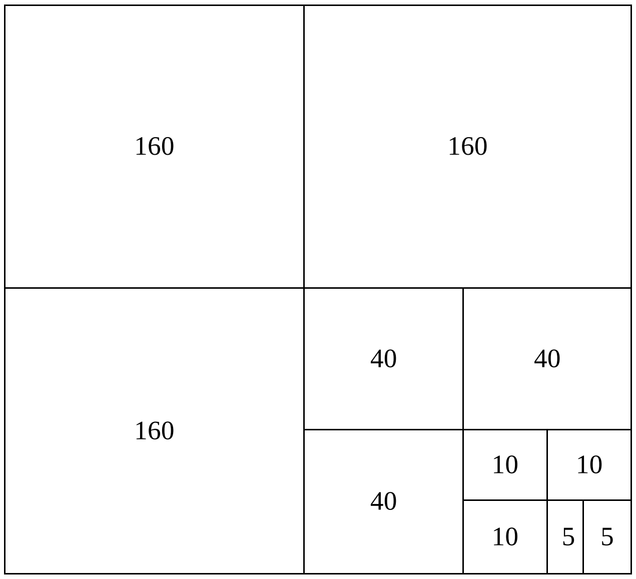
Example Output to Screen

```
7%
0%
```

9. Fences

Program Name: Fences.java Input File: fences.dat

Rob is a surveyor who is plotting the homesteads in a section of land in an unmapped area of Alaska. A section of land covers one square mile and contains 640 acres of land. Rob has divided the section he is surveying into 128 horizontally or vertically contiguous, congruent, rectangular plots of land each containing 5 acres. Below is a diagram of a 640 acre section of land that has been subdivided as noted in the diagram and chart below.



Name	Acres	Shape
Section	640	Square
Quarter section	160	Square
Quarter quarter section	40	Square
¼ quarter quarter section	10	Square
Plot of land	5	Rectangular

To visualize his survey on paper, Rob has created a section grid that is 8 rows by 16 columns. Each cell of the grid represents one 5 acre plot of land that is 660' x 330', that is, on the section grid the vertical sides of the homestead plot are 660 feet long, and the horizontal sides of the homestead plot are 330 feet long. Some people already have homesteads that contain one or more contiguous plots. Rob decided to denote those plots with capital letters of the alphabet using a unique letter of the alphabet for each person. You are to write a program that will tell how many feet of fencing a given homesteader needs to fence his lot.

Note: If a homesteader's land is contiguous to another homestead's land, each homesteader will need a separate fence for the common property line.

Input

The first line of input will contain a single integer `n` that indicates the number of surveys to follow. For each survey,

- the first line will contain an unknown number of capital letters (separated by a space) that appear in the section grid,
- the following lines contain the section grid for the survey, with each grid containing 8 lines with 16 characters on each line.
- The characters in each row will be a capital letter (A-Z) indicating who owns the plot of land, or an asterisk (*) indicating an un-owned plot of land.

Output

For each section grid, print `SURVEY` *X* where *X* is the number of the survey on a single line. On subsequent lines, print, in alphabetical order, the letter of the homestead owner, a space, and the number of feet of fencing that homesteader will need to fence his land.

(continued on next page)

9. Fences (cont.)

Example Input File

```
2
A R D
****************
***DD******R****
**DDD*****RR****
***********R****
*********AA*****
*********AA*****
*********AA*****
*********AA*****
Z A
****************
***A************
***A************
***A************
****************
********ZZZ*****
****************
****************
```

Example Output to Screen

```
SURVEY 1
A 6600
D 4620
R 5280
SURVEY 2
A 4620
Z 3300
```

10. Odd Couple

Program Name: Odd.java   Input File: odd.dat

Felix and Oscar were unlikely roommates in the television series *The Odd Couple*. Felix, being the fastidious one, placed unique matching letters on the bottom of both shoes of each pair of Oscar's shoes to make it easier for Oscar to keep his shoes arranged in pairs. Knowing when to fight his battles, Felix did not insist that the left shoe be on the left of the matching right shoe but did insist that the shoes be lined up in alphabetical order. In one episode, Felix finds Oscar's shoes neatly arranged in a row on the floor of Oscar's closet but not in alphabetical order. Felix can not stand the lack of order and since this seemed to happen frequently, Felix needs you to write a program that will tell him how to arrange the shoes so they will be in alphabetical order.

Each line of your input file will have an even number of shoes that will indicated by a string of capital letters of the alphabet. You will rearrange the shoes so matching shoes are next to each other and are in ascending alphabetical order.

Input

The first line of input will contain a single integer `n` that indicates the number of sets of shoes to follow. Each of the next `n` lines will contain a single string with an even number of capital letters. Each letter represents one shoe and every shoe will have exactly one match.

Output

For each line of input, you will print how the shoes will be arranged when placed in alphabetical order.

Example Input File

```
3
ACBBAC
AFTTERAFER
AWTZZTWA
```

Example Output to Screen

```
AABBCC
AAEEFFRRTT
AATTWWZZ
```

11. Office Party

Program Name: Office.java   Input File: office.dat

Many offices have office parties during the holiday season where people give gifts to some of their co-workers. Sometimes, it seems that one person may give a lot more than they receive while other people receive a lot more than they give. You are to write a program that will determine the net value of the gifts a person receives. The net value of his gifts is the total value of the gifts he received less the amount of money he spent on gifts and could be a negative number. Since each person has a budget, the total value of the gifts he gives will be divided equally among the people receiving his gifts. Do not round or truncate any of these values.

Input

The first line of input will contain a single integer `n` that indicates the number of offices that we will consider. For each office:

- The first line will be a single integer `m` that indicates the number of people in the office.
- The next `m` lines will contain the following pieces of data, all separated by a space:
    - each person's first name with no spaces,
    - the amount of money he budgeted for the year, and
    - the list of co-workers’ first names for whom he bought gifts.

Output

For each person in the office, you will print his name, a space, and the net value of his gifts rounded to the nearest penny. The list of names for each office should be in alphabetical order and there should be at least one blank line after each office.

Example Input File

```
2
4
Mary 100 Ann Alex John
Alex 75 Mary John
Ann 80 Mary Alex John
John 100 Ann Mary Alex
5
Roger 200 Aaron Amber Lloyd
Drew 250 Aaron Roger Amber Lloyd
Amber 100 Roger Drew Aaron
Lloyd 150 Amber Drew Roger
Aaron 0
```

Example Output to Screen

```
Alex 18.33
Ann -13.33
John -2.50
Mary -2.50

Aaron 162.50
Amber 79.17
Drew -166.67
Lloyd -20.83
Roger -54.17
```

12. Reverse Pals

Program Name: ReversePals.java   Input File: reversepals.dat

There are many numbers that are palindromes. A number that is a palindrome is a number that, when the digits are reversed, the number remains the same. The number 121 is an example of a number that is a palindrome.

Most integers that are not palindromes have an integer that we will call a Reverse Pal that is a palindrome. The Reverse Pal for a number is found by a process that begins by reversing the digits in that number and adding that reverse to the number. If the resulting sum is not a palindrome, take the sum and repeat the above process until the result is a palindrome, the Reverse Pal of the original number.

For example, consider the number 97:

- `97 + 79 = 176`
- `176 + 671 = 847`
- `847 + 748 = 1595`
- `1595 + 5951 = 7546`
- `7546 + 6457 = 14003`
- `14003 + 30041 = 44044`   `//`palindrome found after the 6th iteration

You are to write a program to find the Reverse Pal of a given integer and the number of iterations it took to find its Reverse Pal. Although not all integers have a Reverse Pal, you may assume that each integer in our data set does have a Reverse Pal and that it can be found in 20 iterations or less. The number of iterations is the number of times in the process that you reversed the digits of a number and added the reverse to that number.

Input
The first line of input will contain a single integer `n` that indicates the number of positive integers to follow. Each the following `n` lines will contain a single positive integer less than 1000 that is not a palindrome.

Output
For each integer input, you will print the Reverse Pal for that integer followed by a space and the number of iterations it took to find the Reverse Pal.

Example Input File
```
4
97
125
987
882
```

Example Output to Screen
```
44044 6
646 1
66066 4
1881 2
```