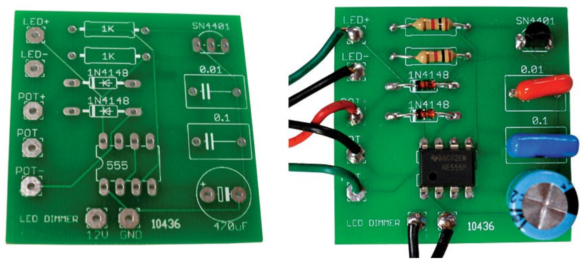
Problem 1: Adjacent Arcs



A local manufacturing firm creates circuit boards and verifies them before shipping. An automated testing machine verifies each circuit by running an ordered list of tests. If the number of tests passed reaches the Circuit Target, the board is packed and shipped, otherwise it is rejected.

Usually the Circuit Target is determined by the number of components. However, some very paranoid customers require a higher Circuit Target. A Critical Component is one with the most direct connections. A Critical Connection is one connected to a Critical Component. The Critical Circuit Target (CCT) is the sum of the number of components and the number of Critical Connections.

Create a program to compute the CCT for a circuit given a list of connections.

The first line of input will be the number of data sets to be processed.

Each data set is described by a line containing the number of connections (0 < N < 1,000,000). Each connection will be a line with two component identifiers (integers, 0 < N < 1,000,000,000) separated by a space.

Output for each data set should be one integer, on a line by itself, the CCT for the circuit.

Sample Input

```
3
5
1 2
2 3
3 1
3 4
2 5
3
1 2
1 3
1 4
4
2 1
3 2
5 4
4 2
```

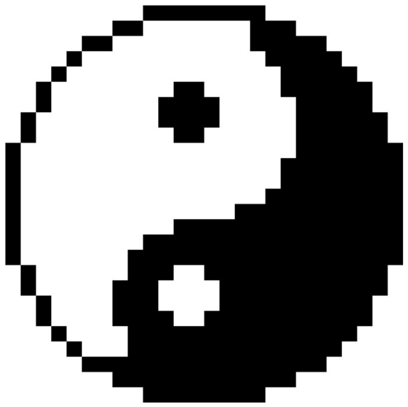
Sample Output

```
10
7
8
```

Problem 2: Bifurcated Bits

There exists a pair of integers (*a* and *b*) for every positive integer *n* known as its Coordinated Congruent Twins (CCT). The CCT for *n* can be found by the following algorithm:

1. represent *n* as a sum of unique powers of 2
2. arrange the powers of 2 in ascending order
3. *a* = sum of the powers of 2 in odd positions
4. *b* = sum of the powers of 2 in even positions



For example, consider the number 45.

1. 45 = 4 + 1 + 32 + 8
2. 1, 4, 8, 32
3. *a* = 1 + 8
4. *b* = 4 + 32

Write a program to find the CCT for positive integers ($n \leq 10^9$).

The first line of input will be the number of integers to be processed.

Each positive integer will follow on a line be itself.

Each integer should produce one line of output with *a* and *b*, in order, separated by a single space as shown below.

Sample Input

```
3
45
7
100
```

Sample Output

```
9 36
5 2
68 32
```

Problem 3: Creepy Characters

A common, yet super-annoying, TV trope known as "All Just a Dream", reveals that the preceeding events were actually all just a dream, when the protagonist wakes up. This gets even more confusing when dreams happen within dreams. To help keep track of complex plots, here at Creepy Characters Theaters (CCT), we need program.

The first line of input will be the number of lines to be processed, less than 100,000.

Each of these lines contains a command followed by an event name. Commands will be one of: **`event`**, **`dream`**, or **`query`**. Event names will contain only lowercase letters (**`a`** thru **`z`**) and underscore (_), and will be at most 50 characters long.

The **`event`** command indicates that a specific named event occurred in the plot. It is guarenteed that this specific event has not yet occurred, unless it was previously part of a dream, in which case it can occur again.

The **`dream`** command indicates that all events back to to specified event where really part of a dream and did not actually occur. If the specified event does not exist, it means all events occurred within a dream.

The **`query`** command is a request to determine whether the specified event actually occurred, as far as we know, at this point in the plot. For each query command, print one line. If the event has previously occurred and is not known to be part of a dream print **`real`**, or else print **`fake`**.

Sample Input

```
12
event normal_stuff
event plot_complication
event joe_dies
query plot_complication
event apocalypse
query joe_dies
dream plot_complication
query apocalypse
event world_war_iii
event apocalypse
query weird_stuff
query apocalypse
```

Sample Output

```
real
real
fake
fake
real
```

Problem 4: Deceptive Dates

To help protect customers from purchasing spoiled food, the Corrupted Cuisine Time (CCT) standard was created in 1980 and requires all foods being sold to be labeled with a "best sold before" date. In any case, this date is commonly know as a CCT, and requires grocers to discard items once the CCT has passed. Apparently no one cares when you actually consume it, just when it is sold.

A CCT consists of three integers, representing the month, the day and the year. Since printing is expensive, years may be represented by only two digits; in this case the first two digits are assumed to be either 19 or 20, providing coverage for 100 years from inception of the standard (*i.e.*, 1980 to 2079).

A shady grocer has observed that while all foods must have a CCT, there is no standard for the order of month, day and year. As long as the CCT can possibly be interpreted as valid, and it is still in the future, he is not required to discard the product. For example, `09.02.07` could mean July 2, 2009 or September 7, 2002 or several other dates.

Similarly, `10/21/19` can be interpreted as October 21, 2019 or October 19, 2021. However, it cannot be interpreted as a date in 2010 because there is no month numbered 21 or 19.

Create a program to compute all possibles dates for any CCT. Recall that a year is a leap year (has 366 days) if the year is divisible by 4, unless it is divisible also by 100, but not by 400 (so 2000 is a leap year, 2100 is not a leap year, and 2012 is a leap year).

The first line of input will be the number of data sets to be processed.

Each data one CCT on a line by itself. While the standard does not require it, all integers in each CCT will be separated by slashes (`/`).

Output should be a list of all possible dates for that CCT, in chronological order and separated by a single space, in the form: `YYYY-MM-DD`. That is, the full four-digit year and exactly two-digits for each month and date, separated by a dash (`-`), as shown below.

Sample Input

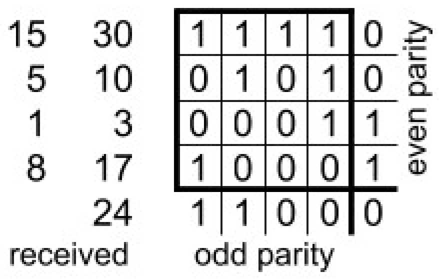
```
5
2/4/67
31/9/30
2/28/29
2/2/02
03/04/5
```

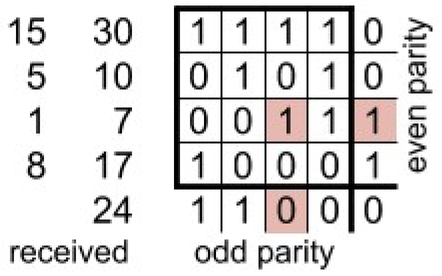
Sample Output

```
2067-02-04 2067-04-02
2031-09-30
2028-02-29 2029-02-28
2002-02-02
2003-04-05 2003-05-04 2004-03-05 2004-05-03 2005-03-04 2005-04-03
```

Problem 5: Erratic Errors

The Correct Communication Transport (CCT) protocol provides a method for transmitting data that allows detecting and correcting one-bit errors using *parity bits*. To transmit data if is first arranged into groups of bits forming a square. Next, one extra bit is added to each row and column (the extra corner bit is always zero). Rows use even parity, which means that including the parity bit, each row should always have an even number of bits. Thus value of the parity bit is determined by the number of ones in that row of the original data. Columns use odd parity, but otherwise are similar.





The example at the right shows the original data (15, 5, 1, 8) and the transmitted/received data (30, 10, 3, 17, 24) after adding parity bits.

If an error of only one bit is made during transmission, the location of the error can determined and the value corrected if necessary. For example, the second grid shows a single bit error in transmission, the third number received was a 7—off by one bit from the original value 3. This causes parity errors in the third row and second column. Knowing this, the bit can be corrected, in this case the one is changed to a zero which both restores the original data and corrects the parity errors.

Create a program to determine the original data given the received data, which *might* include a 1-bit error.

The first line of input will be the number of data sets to be processed.

Each data set begins with a decimal integer (2 ≤ N ≤ 60) indicating the number of rows in the data received. Each row is represented by a decimal integer whose binary value is the row data. All numbers are separated by a single space.

Output should be one line for each data set with decimal integers whose value represents the original transmitted data. Separate numbers by a single space.

Sample Input

```
4
5 30 10 3 17 24
5 30 10 7 17 24
9 257 129 65 33 16 9 5 3 16
9 257 129 65 33 17 9 5 3 16
```

Sample Output

```
15 5 1 8
15 5 1 8
128 64 32 16 8 4 2 1
128 64 32 16 8 4 2 1
```

Problem 6: Fancy Folds

Here at Creative Corner Toys (CCT), every delightful purchase is snuggly packed in a rectangular box and cheerfully wrapped before being gently handed to our satisfied customers by industrious elves. However, lately we have noticed the elves have a tendency to be, shall we say, be "generous" when it comes to the amount of wrapping paper and ribbon they use. In fact, their exuberance is so great we can even sometimes lose money due to the cost of excessive wrapping.

After a careful and extensive analysis, CCT packaging experts have determined that all gifts can safely, equitabily, and respectfully be wrapped by simply calculating the surface area of the box and adding the surface of the smallest face. Elves measure distances in *knuckles*, which are abbreviated "k". For example, a box that measures 2k × 3k × 4k has a total surface area of 2×(6k^2 + 12k^2 + 8k^2). The area of the smallest face is 6k^2, therefore the total amount of wrapping paper required is 58k^2.

Similar painstaking research at CCT has determined that an appropriate bow for any box can be creatively and reliably spun from a ribbon whose length in knuckles is equal to the volume of the box, in k^3. In addition, the ribbon should encircle the shortest dimension of the box. For example the box above needs 24k of ribbon for the bow and 10k of ribbon to encircle the box for a total of 34k.

Please help CCT out by writing a program to compute the minimum wrapping paper and ribbon needs for each box.

The first line of input will be the number of data sets to be processed.

Each data set is described by a line containing the length, depth, and height, in knuckles, for a box. All dimension will be positive integers less than 1000.

Output should be one line for each box. Each line should contain exactly two integers separated by space. The first indicating the amount of wrapping paper needed, measured in number of k^2, the second indicates the length of ribbon needed, again measured in knuckles.

Sample Input

```
3
2 3 4
1 1 5
2 5 7
```

Sample Output

```
58 34
23 9
128 84
```

Problem 7: Genetic Grids

$$\begin{bmatrix} a_{1,1} & a_{2,1} & \cdots & a_{i,1} & \cdots & a_{n,1} \\ a_{1,2} & a_{2,2} & \cdots & a_{i,2} & \cdots & a_{n,2} \\ \vdots & \vdots & \ddots & \vdots & & \vdots \\ a_{1,j} & a_{2,j} & \cdots & a_{i,j} & \cdots & a_{n,j} \\ \vdots & \vdots & & \vdots & \ddots & \vdots \\ a_{1,n} & a_{2,n} & \cdots & a_{i,n} & \cdots & a_{n,n} \end{bmatrix}$$

The Center for Cheap Transformations (CCT) recently discovered a complete set of atomic 2D-maxtrix transformations for square matrices. CCT published a standard nomenclature shown below.

The **identity** transform (i) does not change its input.

The **rotate right** transform (r) rotates the matrix clockwise about its center.

The **rotate left** transform (l) rotates the matrix counter-clockwise about its center.

The **rotate half** transform (u) rotates the matrix 180° about its center.

The **horizontal mirror** transform (h) flips the matrix vertically (top becomes bottom).

The **vertical mirror** transform (v) flips the matrix horizontally (left becomes right).

The **transpose** transform (t) flips the matrix around its primary diagonal (bottom left becomes top right).

The **anti-transpose** transform (a) flips the matrix around its secondary diagonal (top left becomes bottom right).

All possible 2D matrix transformations can be created by combining the atomic transforms. A series of atomic transformations is known as a CCT sequence and is written using the lower-case letters above. For example, `rti` means a matrix should be rotated clockwise to the right, then flipped around its primary diagonal, and finally remain unchanged. For a matrix of size N with rows and columns numbered 1..N, this means an element at (1,1) moves to (N,1), then it moves to (1,N), and finally remains at (N,1). Similarly, an element at (1,N) moves to (1,1).

Obviously, CCT sequences are not unique. In fact there are an infinite number of ways to write the same matrix transformation using CCT sequences. Create a program to determine if two CCT matrix sequences are identical.

The first line of input will be the number of data sets to be processed.

Each data set consists of two lines containing CCT matrix sequences to be compared, each on a line by itself. No line will be longer than 100 characters.

Output should be one line per data set indicating whether the sequences are identical or not, as shown below.

Sample Input

```
3
rti
iiriitii
rr
tt
lta
r
```

Sample Output

```
identical
different
identical
```

Problem 8: Hyper Hedges

Garden mazes have been commissioned by the wealthy for hundreds of years. They delight guests who spend hours wondering the paths in an attempt to find their way to an exit. The Committee on Circuitous Turnings (CCT) recommends warning guests by announcing the minimum number of steps required to reach the exit. Usually this is quite straight forward but a new development has left garden maze owners in a crisis—they need to recalculate the minimum number of steps since Hyper Hedge Hoppers have invaded.

Hyper Hedge Hoppers, a previously unknown species, have taken up residence in all garden mazes, which in itself is not so bad, but they always occur as matched twins and invariably teleport guests to thier twin. Fortunately, Hyper Hedge Hoppers are easy to identify on maps.

Create a program to compute the minimum number steps thru a maze, given the Hyper Hedge Hopper twin teleporting behavior.

The first line of input will contain the number of data sets to process.

Each data set begins with a line containing the width and height of the maze, with a single space between them, both at most 100.

Each row of the maze is given by a line of characters. Period (`.`) indicates the space is open. A hedge is shown as **`X`**. Hyper Hedge Hoppers are indicated by lowercase letters. Any lowercase letters will appear exactly two times in a given maze.

All edge locations will be maize-filled, except one entrance, on the left side, and one exit, on the right side.

Each data set should produce one line of output containing the minimum number of steps needed to reach the goal, if it is possible. If it is not possible to reach the exit, output **`impossible`**.

Sample Input

```
3
10 5
XXXXXXXXXX
X..a.....X
........aX
X.........
XXXXXXXXXX
10 5
XXXXXXXXXX
X.....XbaX
...X.XXXXX
X..XaXb...
XXXXXXXXXX
6 3
XXXXXX
.aXaX.
XXXXXX
```

Sample Output

```
7
12
impossible
```