Problem 1 – CSS

Professor Plum likes it when the College of Saint Scholastica, CSS, hosts MICS. He wants you to write a program to generate an ASCII art "CSS" sign. He plans on taping the sign on the side of the van when traveling to MICS. Since he does not know the dimensions of the sign, he wants your program to take as input positive integer scaling factors and generate multiple signs of different sizes.

Scaling Factor	Letter Dimension of CSS (# chars × # chars)	Line Width of All Letters (# chars)	Blanks Between CSS letters
1	5 × 5	1	5
2	10 × 10	2	10
3	15 × 15	3	15
10	50 × 50	10	50

Input

The first line contains the number of scaling factors which will be an integer between 1 and 100. Each of the following lines contains a single positive integer between 1 and 50 which is the scaling factor. The below sample input has 2 scaling factors.

```
2
1
3
```

Output

The output should contain the ASCII art for each sign corresponding to the scaling factors specified by the input. **NOTE**: All lines for a sign should be the same length by padding shorter lines with blanks. Five blank lines are after each case. Output for the above input is shown below.

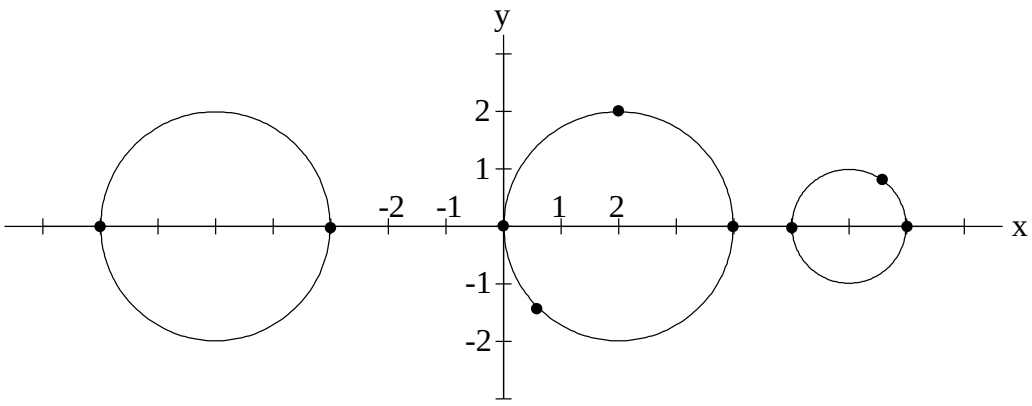
```
Case 1:
CCCCC     SSSSS     SSSSS
C         S         S
C         SSSSS     SSSSS
C             S         S
CCCCC     SSSSS     SSSSS



Case 2:
CCCCCCCCCCCCCCC               SSSSSSSSSSSSSSS               SSSSSSSSSSSSSSS
CCCCCCCCCCCCCCC               SSSSSSSSSSSSSSS               SSSSSSSSSSSSSSS
CCCCCCCCCCCCCCC               SSSSSSSSSSSSSSS               SSSSSSSSSSSSSSS
CCC                           SSS                           SSS
CCC                           SSS                           SSS
CCC                           SSS                           SSS
CCC                           SSSSSSSSSSSSSSS               SSSSSSSSSSSSSSS
CCC                           SSSSSSSSSSSSSSS               SSSSSSSSSSSSSSS
CCC                           SSSSSSSSSSSSSSS               SSSSSSSSSSSSSSS
CCC                                        SSS                           SSS
CCC                                        SSS                           SSS
CCC                                        SSS                           SSS
CCCCCCCCCCCCCCC               SSSSSSSSSSSSSSS               SSSSSSSSSSSSSSS
CCCCCCCCCCCCCCC               SSSSSSSSSSSSSSS               SSSSSSSSSSSSSSS
CCCCCCCCCCCCCCC               SSSSSSSSSSSSSSS               SSSSSSSSSSSSSSS
```

Problem 2 – Circus Sort

Professor Plum has fond memories of visiting a 3-ring circus as a child. When his grandson was learning about the x-y coordinate plane, he promised a trip to the circus if his grandson could complete the following challenge. Given a set of (x, y) coordinates corresponding to points on three rings (i.e., circles), determine which points belongs to which ring and order the points within each ring clockwise from the left-most point. Professor Plum guarantees that:

- the rings don't overlap,
- the ring centers are on the x-axis
- the set of points include all points where the rings intersect the x-axis.

Consider, the following three rings with points shown:



The set of points in no particular order would be:
(4.0, 0.0), (-3.0, 0.0), (-7.0, 0.0), (2.0, 2.0), (5.0, 0.0), (0.0, 0.0) , (7.0, 0.0), (0.5, -1.3), (6.5, 0.9)
Professor Plum wants you to write a program to solve this challenge.

Input

The first line contains the number of points to sorts from the three rings. Coordinates will be rounded to the nearest tenth. The second line contains pairs of floating point numbers corresponding to (x, y) points. For the above example, the input could be:

```
9
4.0 0.0 -3.0 0.0 -7.0 0.0 2.0 2.0 5.0 0.0 0.0 0.0 7.0 0.0 0.5 -1.3 6.5 0.9
```

Output

Three lines of output corresponding to points belongs to each ring from left-to-right. The order of the points within each ring should be listed clockwise starting from the left-most point. The format of the output is shown below including a single space before each point and a single decimal place for each x and y value.

```
Ring 1: (-7.0,0.0) (-3.0,0.0)
Ring 2: (0.0,0.0) (2.0,2.0) (4.0,0.0) (0.5,-1.3)
Ring 3: (5.0,0.0) (6.5,0.9) (7.0,0.0)
```

Problem 3 – Invisible Ink

Professor Plum has a hard time remembering all of his passwords. He decides to store all of his passwords in a text file. To prevent someone from opening the text file and viewing his passwords he encrypts the file using only the white-space characters of blank-spaces (' ', ASCII character 32_{10}) and horizontal-tabs ('\t ', ASCII character 9_{10}). Thus, someone opening the file will see only a blank screen.

Every 7 space/tab characters in the file encodes a binary number where spaces represent 0s and tabs represent 1s. Each 7-bit binary number encodes for an ASCII value between 0 - 127. (NOTE: ASCII and UNICODE values are equal in this range)

Professor Plum has written the program to encrypt the passwords to a text file containing only spaces and tabs. He wants you to write the program to decrypt this file back to the characters for the passwords.

Input

The input contains a single line containing only a multiple of 7 spaces and tabs, except for the ending new-line character. For example the following input (where a space is shown as '`s`' and a tab is shown as a '`t`') encodes the string "Hi Bob!". (ASCII value: 'H' is 72_{10} or 1001000_2, 'i' is 105_{10} or 1101001_2, ..., '!' is 33_{10} or 0100001_2)

```
tsstsssttstsstststsssssstsssststtstttttttsssstsstssssst
```

Output

The output contains only the decrypted characters corresponding to the input. For the above example, the output would be:

```
Hi Bob!
```

Problem 4 – Circular Primes

Professor Plum's doctor thinks he should lose weight, and his students think he is old. However, Professor Plum prefers to think of himself as *circular prime*. A *circular prime* number is one that remains a prime number after repeatedly relocating the first digit of the number to the end of the number. For example, 197, 971, and 719 are all cirular prime numbers. Other numbers that satisfy the circular prime definition are: 5, 11, 13, 37, 79, 113, 199, and 3119.

He wants you to write a program that finds all circular prime numbers between two given positive integers (inclusive to the numbers given). You may assume that both integers are in the range 1 to 500000.

Input

A single line of input containing two integer values between 1 and 500000.

```
5 50
```

Output

The output will be an ascending list of all circular prime numbers between the two integer inputs. The output should have one number per line. For the example input given above, the output is:

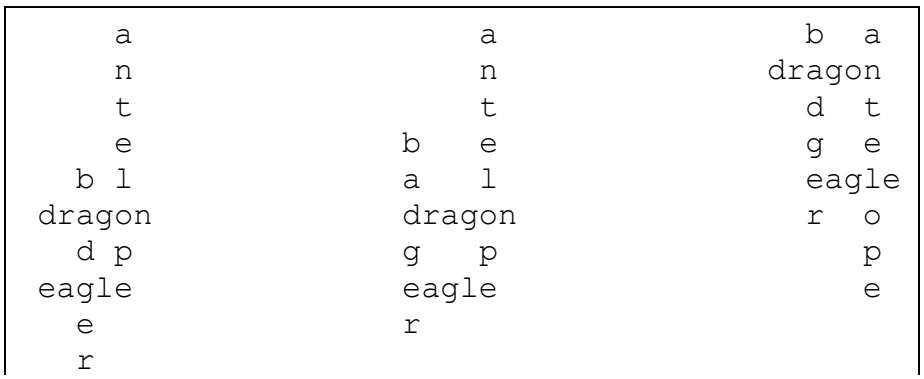
```
5
7
11
13
17
31
37
```

Problem 5 – Wreck Tangles

Professor Plum occasionally teaches Operating Systems, and this problem reminds him of the opposite of deadlock.

Bad news. Four strings all pulled up to a four-way stop at exactly the same time. None yielded, and all four mutually collided with each other and are in a pile-up. Interestingly, they overlap each other only where they have common letters. For example, suppose dragon is the west string, antelope the north, eagle the east, and badger the south. These four might pile up in several ways:

Given any four strings, in how many possible arrangements may they pile up? The four strings must enclose at least one empty square. (The pile-ups above enclose 1, 3, and 4 empty squares, respectively.)



Input

The input consists of a number of cases followed by a sequence of four strings for each test case. The order of the strings is west, north, east, and south.

```
2
dragon
antelope
eagle
badger
rabbit
tiger
t-rex
ant
```

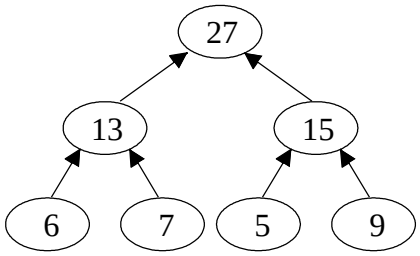
Output

For each case, print a case label and the number of possible pile-ups. For the above input, the output is:

```
Case 1: 3
Case 2: 1
```

Problem 6 – Summit

Professor Plum likes vacationing in the mountains. He wants you to consider this mathematical mountain, where each non-leaf node is the sum of its two children:



Something's wrong, however. $27 \neq 13 + 15$, and $15 \neq 5 + 9$. If we replace the 15 with 14, then we have a well-formed mathematical mountain. Given a serialized version of a mountain, where the root is element 0, the root's left and right children are elements 1 and 2, and so on down the mountain, print the index of the single incorrect element and the corrected value. If the incorrect value occurs on the leaf level, then the right child is assumed to be wrong.

Input

The input consists of a number of cases followed by a line for each tree. The first number on a mountain's line is the number of levels in the mountain. The remaining numbers are the node values, separated by whitespace and in breadth-first order. A mountain will always have at least 3 levels. Each mountain is full and complete, meaning that all non-leaves have exactly two children and that all leaves are on the bottom-most level.

```
3
3 27 13 15 6 7 5 9
4 21 9 10 4 5 4 6 2 2 1 4 1 3 2 4
3 29 13 16 5 8 9 1
```

Output

For each case, print a case label, the index of the incorrect node, and the correct value. For the example input, the output is:

```
Case 1: 2 14
Case 2: 0 19
Case 3: 6 7
```

Problem 7 – The Plot Thickens

Professor Plum's wife likes to paint, but Professor Plum is more of a digital kind of guy. Imagine an 8-by-7 canvas of zeroes as shown in Figure 1a. Imagine plotting a 5-by-4 rectangle of ones on it, with its top-left corner at (2, 3), as shown in Figure 1b.

00000000 00000000 00000000 00000000 00000000 00000000 00000000	00000000 00000000 00000000 00111110 00111110 00111110 00111110	00000000 11110000 11110000 11001110 11001110 11001110 00111110	00000000 11110000 11101000 11010110 11001110 11001110 00111110
(a) Blank canvas	(b) First rectangle	(c) Second rectangle	(d) Third rectangle

Imagine plotting a 4-by-5 rectangle of ones on the current canvas, with its top-left corner at (0, 1). However, whenever the rectangle overlaps any other rectangle, the pixels cancel each other out, as shown in Figure 1c. Suppose further that we plot a 2-by-2 rectangle at (3, 2). The resulting canvas is shown in Figure 1d.

After plotting a sequence of rectangles to a canvas in this manner, how many pixels are set to 1?

Input

The input consists of a number of cases followed by a line for each case. The first two numbers in each case's line are the width and height of the canvas. The third number is the number of rectangles plotted. The remaining numbers describe each rectangle and therefore appear in groups of 4. Within a group, the first two numbers are the xy-coordinates of a rectangle's top-left corner, and the second two are the rectangle's dimensions.

```
2
8 7 3 2 3 5 4 0 1 4 5 3 2 2 2
3 3 1 0 0 3 3
```

Output

For each case, print a case label and the number of 1-pixels in the canvas after all plotting. For the example input, the output is:

```
Case 1: 28
Case 2: 9
```

Problem 8 – Highly Recursive Function

Professor Plum likes recursion, but his students typically find it confusing. During a recent faculty meeting his mind wandered, and he invented the following recursive mathematical function, H(n):

$H(n) = H(n+5) + H(n+4) + H(n+2)$ for all value of $n \leq -8$

$H(n) = n$ for all value of $-8 < n < 10$

$H(n) = H(n-8) + H(n-5) + H(n-3)$ for all values of $n \geq 10$.

He wants you to write a program to compute values of the function H(n).

Input

The first line contains the number of n values to run through the function H(n). Each of the following lines contain a single integer value of n. All of the values of n and corresponding H(n) values will fit into a 64-bit signed integer. The below sample input contains three n values.

```
4
-8
10
-13
-4
```

Output

For each n value, print to standard output a case label and the value of H(n) as defined above. For the example input given above, the output is:

```
Case 1: H(-8) = -13
Case 2: H(10) = 14
Case 3: H(-13) = -58
Case 4: H(-4) = -4
```