Problem 1

Violent Numbers

Mr. Drump is afraid of the growing terrorism and wants some serious counter measures. His country's think tank reported to him that a person could possibly be a terrorist if his/her date of birth (DoB) could be approached by violent numbers. You're required to help Drump to identify terrorists.

We say, DoB could be approached if it has at-least ***X*** violent numbers (from 1 to DoB's integer equivalent) where ***X*** is the day value of DoB. Let D represents DoB's integer equivalent, and Y is any number from 1 to D, then we say Y is violent if:

- Sum of **D**'s even digits is a factor of ∑n where n is even and have values in range $0<n \leq Y$
- Let **Z** = ∑n where n is odd and have values in range $0<n\leq Y$, **Z** is funny and divisible by the sum of odd digits of **D**

We say, a number is funny who is divisible by at-least five integers excluding 1 and number itself.

Input

The input consists of multiple test cases. The first line of input is the number of test cases N where (0<N<5000). Each of the following N lines contains a DoB as **Day**<space>**Month**<space>**Year** where valid year would be 1 to 10^{10}. For day and month assume there would be valid values. For example, 18-Aug-1995 would be 18 8 1995 and its integer equivalent is 1881995.

Output

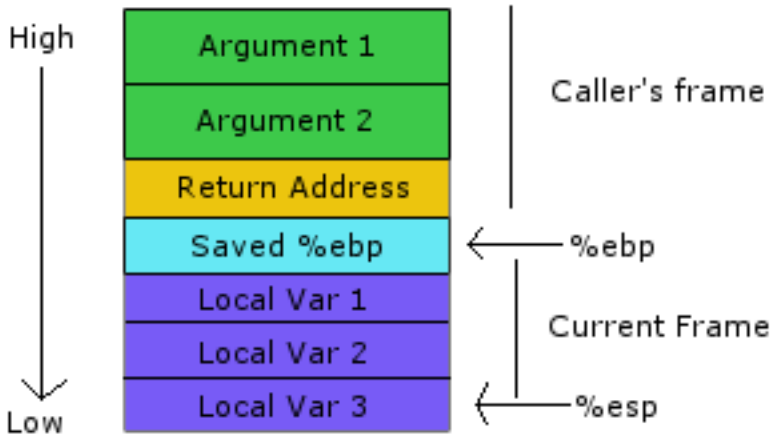
For each test case, print a single line that says "Case**N**:" where **N** is the test case number followed by the text "Terrorist Found" if the given DoB could be approached, print "Peace" otherwise.

Sample Input	**Sample Output**
4 1 1 1 1 1 2 5 2 7 23 7 99	Case1:Peace Case2:Terrorist Found Case3:Terrorist Found Case4:Terrorist Found

Problem 2

Malicious Strings

Hackers use number of ways to hijack the execution flows to be able to execute their malicious code. One of these hijacking methods is buffer overflow. A buffer is overflowed when it's filled with more data than buffer's size. Buffer overflow attack finds a buffer, in local variables of a function, which could be overflowed and then prepares a string in such a way that could load the malicious code's starting address in place of return address (RA) in stack frame of the current function. So, when the function returns; instruction pointer (IP) gets loaded with malicious code's address eventually succeeding the hacker. Following is a typical x86 stack frame.



In this problem you'll be given some strings. Each string could contain multiple malicious sub- strings and you're required to count these ones. A string is malicious if it contains a valid address that could possibly be loaded in place of **RA**. An address is valid if it starts with "**0x**", only contains hex digits and meets the system architecture i.e. 8bit system's address would be of 1 byte, 16bit's would be of 2 bytes and so on. Valid hex digits are 0-9 and A-F (case insensitive).

Consider each character of string equal to one byte in size except hex digits **in a valid address** which would be of half byte. Please also note that **0x** would just behave as an escape sequence of valid address and equals to zero bytes **in case of valid address** otherwise two bytes.

While looking for valid addresses take the non-greedy approach. For example, string "**0x12A0x1F**", in a 16-bit architecture, should be read having two parts of "0x12A" and "0x1F", not as, "0x12A0" and "x1F".

Input

The input consists of multiple test cases. The first line of input is the number of test cases **N** where (0<**N**<5000). Each of the following **N** lines contains an integer, **K**, representing system's architecture and could be 8, 16, 32 and so on up to 2^{10}. This is followed by a string.

Output

For each test case, print a single line that says "Case**N**:" where **N** is the test case number followed by count of malicious sub-strings found in that string. If the count is zero print "Clean String".

Sample Input	**Sample Output**
4 8 xyBsheu0xAuo12334%83462Erfh 8 jhd720x63@#&0xAAA273t0xAkwg@#^2qeh0x12 16 jhd720x63@#&0xAAA273t0xAkwg@#^2qeh0x120x1F 16 sh%rds56E^4s0xFFFF0xAAAAopop0x1234	Case1:Clean String Case2:3 Case3:1 Case4:2

Hint

In a malicious sub-string; there must be **Z** bytes before a valid address that would be written over memory area of buffer-overflowed and saved EBP. Hence **Z** >= 1 + **K**/8.

Problem 3

Beautiful Face

Tania is getting married soon. She has many wishes to get true, one of which is to look beautiful enough. You're required to help Tania in calculating her beauty, so, she can take necessary steps to meet her beauty wish accordingly.

You'll measure her beauty with reference to following ideal face represented in a 15x15 grid

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 3 5 3 0 0 0 3 2 1 0 0 0
0 0 0 5 3 5 0 0 0 4 5 6 0 0 0
0 0 0 3 0 3 0 0 0 7 8 4 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 2 0 0 0 0 0 0 0
0 0 0 0 0 0 0 3 0 0 0 0 0 0 0
0 0 0 0 0 0 0 5 0 0 0 0 0 0 0
0 0 0 0 0 0 0 8 0 0 0 0 0 0 0
0 0 0 0 0 0 0 13 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 3 5 8 13 21 34 55 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

and broken down as following:

- Left eye is a **3x3** grid whose elements sum to 30.
- Right eye is also a **3x3** grid whose elements sum to 40.
- Nose is a **1x6** grid whose elements represent a Fibonacci series.
- Lips are a **1x7** grid whose elements also represent a Fibonacci series.

Tania's face will also be represented in a **15x15** grid and beauty will be a floating-point value calculated as:

A = cosine similarity of left eyes
B = cosine similarity of right eyes
C = cosine similarity of noses
D = cosine similarity of lips

Beauty = A + B + C + D

A, B, C and D could be floating-point values, truncate them up-to four decimal places, if needed.

Cosine similarity could be computed of vectors not matrices. Nose and lips are already vectors but eyes need a conversion. A **3x3** eye would constitute a **1x9** vector whose first three elements will be first row of eye; middle three will be second row resulting last three equal to third row.

Here is the formula to calculate cosine similarity:

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|_2 \|\mathbf{B}\|_2} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$$

where **0<= A_i , B_i <=5000** are components of vector **A** and **B** respectively.

The resulting similarity ranges from **−1** meaning exactly opposite, to **1** meaning exactly the same, with **0** indicating orthogonality, and in-between values indicating intermediate similarity or dissimilarity.

The Fibonacci series is a sequence, called the Fibonacci sequence, characterized by the fact that every number after the first two is the sum of the two preceding ones:

0, 1, 1, 2, 3, 5, 8, 13, 21, ...

There is another headache; in Tania's face grid; eyes, nose and lips are not placed as of ideal grid but on random locations, nose could be horizontal and lips vertical. So, first you need to search these parts, in 15x15 grid, then find the similarities. To make the matter simple, let's assume there will be no duplications, exactly one 3x3 grid sums to 30, exactly one to 40, exactly one pattern builds nose and exactly one lips. Further assume there will be no overlaps of these four parts in grids.

Input

The input consists of multiple test cases. First 15 lines of input represent the ideal face grid. Next line of input is the number of test cases **N** where (0<N<5000). Each of the following 15 lines will represent a grid separating each value with space.

Output

For each test case, print a single line that says "Case**N**:" where **N** is the test case number followed by "Beautiful" if **beauty** is greater than **3.3000** otherwise print "Need Cosmetics". While comparing real numbers an error up-to **0.0010** is tolerated and acceptable.

In a given **15x15** test grid there may be no eye(s), nose or lips. In that case print "Lips" if lips are not present, print "Nose" for missing nose, print "Left Eye" for left and "Right Eye" for right eye missing, print "Eyes" if both eyes are missing, print "No Face" if all of the parts are missing. Apart from these above mentioned cases, if multiple parts are missing then print them alphabetically in title case (first character in caps) separating by commas (,) using Nose, Lips, Left Eye and Right Eye. For example, print "Lips, Nose" if nose and lips are missing and print "Left Eye, Lips" if left eye and lips are missing.

	Sample Input	Sample Output
Reference Face	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 **3 5 3** 0 0 0 **3 2 1** 0 0 0 0 0 0 **5 3 5** 0 0 0 **4 5 6** 0 0 0 0 0 0 **3 0 3** 0 0 0 **7 8 4** 0 0 0 0 0 0 0 0 0 0 **1** 0 0 0 0 0 0 0 0 0 0 0 0 0 0 **2** 0 0 0 0 0 0 0 0 0 0 0 0 0 0 **3** 0 0 0 0 0 0 0 0 0 0 0 0 0 0 **5** 0 0 0 0 0 0 0 0 0 0 0 0 0 0 **8** 0 0 0 0 0 0 0 0 0 0 0 0 0 0 **13** 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 **3 5 8 13 21 34 55** 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Case1:Beautiful Case2:Need Cosmetics
Test 1	2 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 13 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 2 1 0 0 0 0 0 0 0 0 0 0 0 0 4 5 6 0 0 0 0 0 0 0 0 0 0 0 0 7 8 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 5 8 13 21 34 55 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 5 3 0 0 0 0 0 0 0 0 0 0 0 0 5 3 5 0 0 0 0 0 0 0 0 0 0 0 0 3 0 3	
Test 2	0 0 0 0 0 0 0 0 0 0 0 1 30 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 21 34 55 89 144 233 377 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 10 0 10 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 377 610 987 1597 2584 4181 0 0 0 0 0 0 0 0 0	

Problem 4

Neighbor Identification

Central city has a unique home arrangement. Everyone wants to live near to center. Therefore, each new home is built around the boundary of last build homes. See arrangement of 81 homes for better understanding; however; there can be 2^{63} homes.

50	51	52	53	54	55	56	57	58
81	26	27	28	29	30	31	32	59
80	49	10	11	12	13	14	33	60
79	48	25	2	3	4	15	34	61
78	47	24	9	1	5	16	35	62
77	46	23	8	7	6	17	36	63
76	45	22	21	20	19	18	37	64
75	44	43	42	41	40	39	38	65
74	73	72	71	70	69	68	67	66

Unfortunately a virus spread in the city and people got infected. Health department wants you to find left, right, up, down neighbors of an infected person for immediate treatment to prevent serious damage.

Input

The first line of input is the number of test cases N (0<N<5000). Each of the following **N** lines contains an integer **K** where (0<K<=2^{63})

Output

For each test case, print a single line that says "Case**N**:" where **N** is the test case number followed by four neighbors in order of left, up, right, bottom

Sample Input	Sample Output
5 21 38 45 80 1234567890123456789	Case1:22 8 20 42 Case2:39 37 65 67 Case3:76 46 22 44 Case4:119 81 49 79 Case5:1234567890123456790 1234567885679012368 1234567890123456788 1234567894567901218

Problem 5

Walk on a Tree

Binary Tree is a useful data structure where each node has at most two children, usually they are distinguished as **left** and **right** child. The node having the children is called parent. An instruction to walk on a binary tree is a string consisting of the letters L and R where **L** stands for **move left** and **R** for **move right.**

In this problem you will simulate a walk on a **completely filled** binary tree of height at most 100. Completely filled means tree has maximum possible nodes according to its height. The walk will always start at the root of the tree and will be specified by an **instruction string S** such that the length of **S** is always less than that of the height of the tree. **The only trick,** while following the instructions in **S**, is that you may **skip any instruction** in the **string S** (possibly discarding all of them).

Your job is to compute the number of nodes the simulation can end up after following the instruction string S.

For example: Suppose: S = LR. Then following S we can end up in **four** different nodes.

1. Skipping all letters: we will be at the root node.
2. Skipping L and following R: we will be at the right child of the root node.
3. Following L and Skipping R: we will be at the left child of root node.
4. Following both L and R: we will be at the right child of the left child of the root node.

Input

First line of the test file contains an integer **N** denoting number of test cases. Hence, follow N test cases. Each test case consists of a number **H** specifying the height of the tree followed by a non empty instruction string **S**.

You may assume that there will not be any letter other than L and R in the string S and that the length of the string S will be less than the height of the tree H.

Output

For each test case print "Case**N**:" where **N** is test case number, followed by the number of nodes we can end up finally. Since the answer may be very large, you have to give the answer modulo **21092013**.

Sample Input	**Sample Output**
2 10 L 100 LR	Case1:2 Case2:4

Problem 6

Protein Synthesis

Professor Abdul is studying the protein synthesis in *Centaurea solstitialis* a plant species otherwise known as Knapweeds in common parlance. Thanks to his rigorous post-doctoral hackery and wizardry Abdul has identified that the synthesis is in fact a linear transformation of all the amino acid compounds consumed in the process; essentially meaning that given a sequence of amino acid compounds the protein synthesis will be identical regardless of the order in which each compound was consumed.

Now as part of his research experiment Abdul wants to conduct a comparative study among different recorded instances of controlled protein synthesis.

Your job is to help write a program that given two sequences of amino compounds, determines whether or not the protein synthesis is going to yield an identical result in both instances.

Input

The input consists of multiple test cases. The first line of input is the number of test cases **N** where (0<N<5000). Each of the following **2N** lines contains N pair of strings to test, each unique character in the string will represent a different type of amino acid compound. Each string will be under 2^{32} characters in length.

Output

For each test case, print a single line that says "Case**N**:", where **N** is the test case number followed by the text "YES" if the two strings will yield an identical protein synthesis otherwise print "NO".

Sample Input	**Sample Output**
3 HILMPSTV HILTSMVP stvlmpih stvlmpii HE%JKHG ERKJG;clk k%KKGHEJlRJ ;EHGc	Case1:YES Case2:NO Case3:YES

Problem 7

Non-Quantum Entanglement

Alt-Corp Multinational (ACM) has created the world's first quantum computer and they are now launching a domain specific language along with a proprietary script execution engine. This product is expected to bring the company a lot of revenue.

Of course, the customers of this product are learned software craftsmen and pragmatic physicists who have mastered the concept of modularization and like to breakdown the problem at hand into a number of sub-programs or scripts that depend on each other in a spectacularly messy fashion, hence upholding the coveted ethos of software reusability.

As you can imagine that each customer uses this product in a way that they have multiple scripts to execute and all these scripts have various dependencies on each other. Now ACM's product owner is losing sleep over how to efficiently determine a "sane-order" in which the user submitted scripts can be executed. From usability point of view customers cannot be expected to submit their scripts in said "sane-order" because they are already too busy disentangling the quantum bits in their heads. So, like all SCRUM projects, this opportunity has befallen on the team's junior most engineer Taimur, who has been tasked to determine such a "sane-order".

Given a list of scripts to execute and their dependency on each other, your task is to help Taimur determine an order in which the list of scripts can be executed while honoring their dependencies.

Input

First line lists the number of test cases **N**, followed by **N** test cases. First line of each test case contains a list of **K** scripts to be executed. Subsequent **K** lines contains comma-separated list of all the scripts that a given script depends on.

For example: Script1=Script2,Script3,Script4
Means that execution of scripts (Script2,Script3,Script4) is a prerequisite for the execution of Script1.
It can be assumed that there will be no circular dependency amongst the submitted scripts.

Output

For each test case, print a single line that says "Case**N**:", where **N** is the test case number, followed by a sane order that scripts can be executed in.
In case of a tie between a pair of scripts: the script that appeared first in the original input list (as found in the first line of each test case) should be given precedence.

Sample Input	Sample Output
2 A,B,C,D,E,F,G,H,I,J A=J B=A C=A,B D=A,B,C E=A,B,C,D F=A,B,C,D,E G= A,B,C,D,E,F H= A,B,C,D,E,F,G I= A,B,C,D,E,F,G,H J= S1,S10,S15,S16,S17,S18 S1= S10=S15,S16 S15=S16 S16=S17,S18 S18= S17=	Case1:J,A,B,C,D,E,F,G,H,I Case2:S1,S17,S18,S16,S15,S10

Problem 8

Nature Retreat

The corporate big-wigs at the Alt-Corp Multinational (ACM) have had the epiphany to instigate a team building exercise for all their departments.

In an effort to fuse the collaborative nature of an open office floor and the traditional serene nature retreat they have decided to conduct this exercise in the beautiful apple orchards of Swat valley.

Each team will have an orchard of their own and their task is to maximize the collection of apples by moving diagonally through the orchard's entry all the way to its exit.

The programming language design division of the ACM is a very competitive bunch and in order to uphold their intellectual superiority over rest of the company, they have decided to hack the exercise and finish it as soon as possible and then go sightseeing in the Swat valley. As usual this requirement has been handed out to the junior most engineer Taimur.

Your job is to help Taimur determine the maximum number of apples that can be picked up in a given orchard under a number of constraints.

You have a rectangular apple orchard full of ripe apples. The orchard is divided into squares and is represented by a grid. Each element in the grid is an integer, showing the number of apples at the corresponding square in the orchard.

Your goal is to collect as many apples as you can by walking from the bottom left corner of the grid to the top right corner, taking all apples in each square along the way. From a given square you can only go either up, or to the right.

For example, consider that this is the orchard:

4	0	1
1	0	0
0	4	0

At most you can collect 6 apples, by going up (+1), up (+4), right (+0), right (+1).

Now to make things even more interesting: you also have **T** tokens. While you are walking in the orchard, you can use the tokens to double the number of apples on the current square. You can only use one token on any given square.

Input

First line contains the total number of tests **N** the first line of each test has the following format: **R**<space>**C**<space>**T**. Where:
R = Number of rows of the orchard (0 < R <= 200)
C = Number of columns of the orchard (0 < C <= 200)
T = Number of tokens that must be used to maximize the collection of apples (T is a non-negative integer and will never exceed the length of the shortest path of orchard)

Next **R** lines contains **C** values in a space separated list, detailing the layout of the orchard

Output

For each test case, print a single line that says "Case**N**:" where **N** is the test case number followed by the maximum number of apples that can be collected having applied the tokens.

Sample Input	**Sample Output**
2 3 3 2 4 0 1 1 0 1 0 4 0 1 3 3 4 1 0	Case1:11 Case2:10

Problem 9

Decode Rolex Fan Code

Rolex Fan is a large fan manufacturing company facing product duplication by some fraudulent companies. They solve the problem by placing **code** on its product. They instructed all shopkeepers to verify the product by confirmation of code from company. Code is designed in following steps:

1- Take random hexadecimal number of *size*, where 3 < size <= 10000	**Example: (Hexadecimal #:** A7BE**)** A 7 B E
2- **Convert** hexadecimal number into binary number	1010 0111 1011 1110
3- **Rearrange** binary number. Pick every binary digit at index divisible by 4 (starting from 0 index) and place at right side of the number	010 111 011 110 1 0 1 1
4- **Group** binary number into set of 3 elements. Start from left side, add 0's to last group, if required	010 111 011 110 101 1 010 111 011 110 101 001
5- Take **One's complement** (toggle/invert each bit)	101 000 100 001 010 110
6- Next make **group** of 4 elements starting from left side, add 0's to last group, if required	1010 0010 0001 0101 10 1010 0010 0001 0101 0010
7- Again take **One's complement** of number	0101 1101 1110 1010 1101
8- Finally convert binary into hex number. It is the final code	5 D E A D **Coded Hexadecimal #: 5DEAD**

Unfortunately they lost the pair of original and coded hexadecimal numbers. However, luckily they found lists of all hexadecimal number picked for decoding in a batch written for uniqueness. They used to manufacture hundreds of thousands fans in each batch, therefore, it is difficult to code all of them for manufacturing. However, they hire you to write a program for decoding. After decoding they can search this code from list easily.

Input

The input has **N** test cases. The first line is the number **N** (0 < **N** ≤5000) that is number of test cases. Each test case has 2 lines. First line has code in hexadecimal. Second line has format: **K**<space>**HexList**. Where:

K = Number of hexadecimal numbers to test (1≤ K ≤50000)

HexList = Space separated list of K hexadecimal numbers

Output

For each test case print "Case**N**:" followed by decoded code if found in list or -1, if there is no matching number.

Sample Input	Sample Output
6 5DEAD 7 5DEAD 10001 A7BE B236A 1DE2FB 23RAB A80F 4BB972FDE 8 4BB971FDE 4B0971F1E 4BB972FDE B236A 4BB9700DE 4BB970FDE 4BB971FDF AAFBCDE2 5DEAD 7 5DEAD 10001 BA7BE B236A 1DE2FB 23RAB A80F 819C65 15 D0C113 1FB52F A5CA21E DC955DAE 482E5C56 AD 4FB3 71B03A6 72B 92B 53 3DCE58 25B0823A C03969 ACE83603 70D664E3D 40 549C28 BA2 D313DC 311E8 426DD8095 F6B 14E3A6 FE 9C74 CCFD5 13642E A4 145234C06 F2CE880BA 1CBF1 6793AF9 CDB F5C0B8 0D144D F51FE 62EBB9C 5EC041A8D 9E FF 932 BF6E62 B5674 B608 C88 B358C28FA 11BB6EC 82 E31E82AEF 159138F 590DC888E 341DB9C4F 4D51 67B10B4 CCE6 5AD37482E 70D664ED3 40 549C28 BA2 D313DC 311E8 426DD8095 F6B 14E3A6 FE 9C74 CCFD5 13642E A4 145234C06 F2CE880BA 1CBF1 6793AF9 CDB F5C0B8 0D144D F51FE 62EBB9C 5EC041A8D 9E FF 932 BF6E62 B5674 B608 C88 B358C28FA 11BB6EC 82 E31E82AEF 159138F 590DC888E 341DB9C4F 4D51 67B10B4 CCE6 5AD37482E	Case1:A7BE Case2:AAFBCDE2 Case3:-1 Case4:C03969 Case5:341DB9C4F Case6:-1

Problem 10

Separation of Chemical Components

A company making detergents has been experimenting a lot of chemical combinations to make their detergent better. They need to perform separation of components during the process. The people at the company were puzzled over the method to separate components, whether they should remove the components one by one or use a different approach. Each approach has its own merits and demerits. Also, sometimes it is not possible to separate out single component due to its nature. Some engineer in the company has decided to use a known heuristic to remove products from the mixture and then further split the product into sub products until we get the separated out component. The heuristic gives the number of possible separation sequences that will help separate sequences from a set of n components and follows the following trend:

1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, …

where each mixture with **n** components can be distilled using the number of sequences represented by the (**n-1**)th element in the above trend. For example (if we start with **n**=0, then 42 in the trend is the 5^{th} element in the trend and 132 is the 6^{th} element) there will be 42 possible separation sequences if we have a mixture of 6 components. This method is based on separation of cheaper components (products) first and the sequences generated are also optimal in terms of cost. The engineer now needs to calculate the possible number of separation sequences given the number of components in a mixture. You have to help the engineer and give a programming solution that takes number of components as input and gives the possible number of separating sequences based on the above trend.

Input

First line gives the number of **T** test cases, where (0<**T**≤50). The next T lines have number of components **n** (1≤**n**≤36) for each case.

Output

For each test case print "Case**N**:" where **N** is the test case number, followed by the possible number of separation sequences.

Sample Input	Sample Output
6 2 3 4 5 6 13	Case1:1 Case2:2 Case3:5 Case4:14 Case5:42 Case6:208012