Problem A: Rubik's Rectangle

A new puzzle which aims to conquer the game market is a fusion of Rubik's Cube and Fifteen. The board is an $H \times W$ frame with tiles with all numbers from 1 to $H \cdot W$ printed on them.

1	2	15	4
8	7	11	5
12	6	10	9
13	14	3	16

The only type of move that is allowed is *flipping* either one of the rows or one of the columns. Flipping reverses the order of the row's (or column's) elements. Below the third row is flipped:

1	2	15	4
8	7	11	5
9	10	6	12
13	14	3	16

You are given a board with tiles numbered in some arbitrary order. Determine a sequence of flips that brings the board to the nicely sorted position, if possible.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Input

The first line of input contains the number of test cases T. The descriptions of the test cases follow:

The description of each test case starts with an empty line. The next line contains two space-separated integers W and H ($1 \leqslant W, H \leqslant 100$) – the width and height of the puzzle, respectively. Each of the next H lines contains W space-separated integers – the numbers printed on consecutive tiles.

Output

Print the answers to the test cases in the order in which they appear in the input. Start the output for each test case with the word **POSSIBLE** or **IMPOSSIBLE**, depending on whether it is possible to solve the puzzle. If a solution exists, print (in the same line) first the number of moves (possibly 0) and then their descriptions, each consisting of a single letter **R** or **C** specifying whether we are to flip a row or a column, concatenated with the index of the row or column to flip.

Any solution will be accepted as long as it does not use more than $10 \cdot W \cdot H$ moves. Each test case is either solvable within this limit, or not solvable at all.

Example

For an example input	a possible correct answer is:
4 3 3 1 2 3 4 5 6 9 8 7 4 2 1 2 3 4 5 6 7 8 4 4 1 2 15 4 8 7 11 5 12 6 10 9 13 14 3 16 3 4 1 2 4 3 5 6 7 8 9 10 11 12	POSSIBLE 1 R3 POSSIBLE 0 POSSIBLE 3 R3 C3 R2 IMPOSSIBLE

Problem B: What does the fox say?

Determined to discover the ancient mystery – the sound that the fox makes – you went into the forest, armed with a very good digital audio recorder. The forest is, however, full of animals' voices, and on your recording, many different sounds can be heard. But you are well prepared for your task: you know exactly all the sounds which other animals make. Therefore the rest of the recording – all the unidentified noises – must have been made by the fox.

Input

The first line of input contains the number of test cases T. The descriptions of the test cases follow:

The first line of each test case contains the recording – words over lower case English alphabet, separated by spaces. Each contains at most 100 letters and there are no more than 100 words. The next few lines are your pre-gathered information about other animals, in the format <**animal**> **goes** <**sound**>. There are no more than 100 animals, their names are not longer than 100 letters each and are actual names of animals in English. There is no **fox goes ...** among these lines.

The last line of the test case is exactly the question you are supposed to answer: *what does the fox say?*

Output

For each test case, output one line containing the sounds made by the fox, in the order from the recording. You may assume that the fox was not silent (contrary to popular belief, foxes do not communicate by Morse code).

Example

For an example input

```
1
toot woof wa ow ow ow pa blub blub pa toot pa blub pa pa ow pow toot
dog goes woof
fish goes blub
elephant goes toot
seal goes ow
what does the fox say?
```

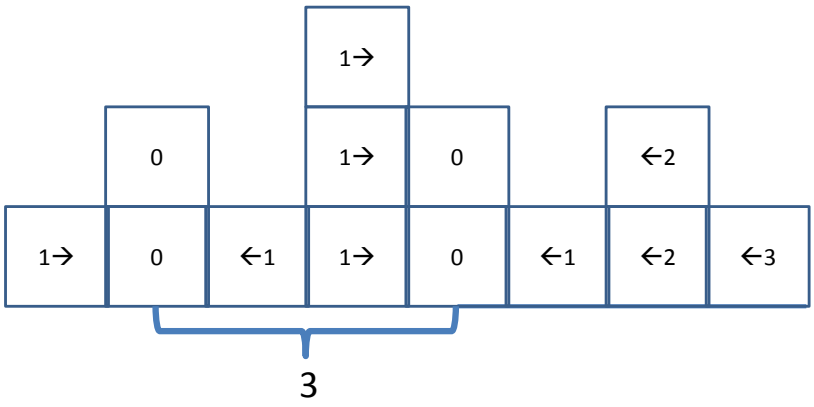
the correct answer is:

```
wa pa pa pa pa pa pow
```

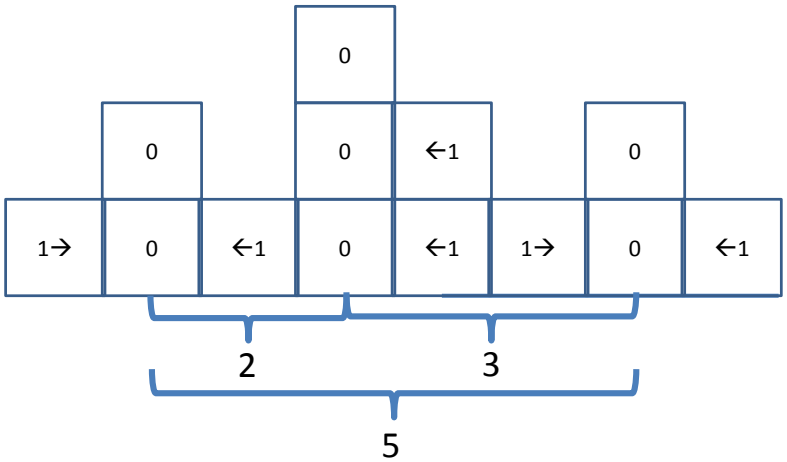
C: Beautiful Mountains

Paco loves playing with stacks of blocks. He likes to pretend that they are mountains, and he loves making his own terrain. Lately, he's been restricting himself to a particular way of rearranging the blocks. He puts all of his stacks of blocks into a straight line. Then, he only changes the arrangement one block at a time. Paco does this by finding two adjacent stacks of blocks and moving one block from one stack to the other.

Paco has made all sorts of arrangements of his 'mountains' using this technique. Now he has decided to make his most beautiful arrangements yet. Paco finds a mountain range beautiful if, for every pair of mountains, the distance between the two mountains is a prime number (that's every pair, not just every adjacent pair). A mountain range with a single stack is beautiful by default. Paco considers a stack of blocks to be a mountain if it has at least one block.



This diagram shows an initial configuration of the blocks, and a way to make two stacks, at a distance of three apart, with 13 moves. However, with only 6 moves, Paco can make a beautiful arrangement with 3 stacks:



Given a current arrangement of blocks, what is the minimum amount of effort needed for Paco to make it beautiful?

Input

There will be several test cases in the input. Each test case will begin with a line with one integer ***n*** (1≤***n***≤30,000), which is the number of stacks of blocks. On the next line will be ***n*** integers ***b*** (0≤***b***≤1,000), indicating the number of blocks in that stack. The integers will be separated by a single space, with no leading or trailing spaces. Note that every stack will be listed, even those with zero blocks. End of input will be marked by a line with a single **0**.

Output

For each test case, output a single integer indicating the least number of moves required to make Paco's stacks of blocks 'beautiful'. Output no spaces, and do not separate answers with blank lines.

Sample Input	**Sample Output**
5 1 2 1 2 1 8 1 2 1 3 2 1 2 1 0	3 6

acm International Collegiate Programming Contest

IBM event sponsor

D: Electric Car Rally

In an attempt to demonstrate the practicality of electric cars, ElecCarCo is sponsoring a cross-country road rally. There are ***n*** charging stations for the rally where cars may check in and charge their batteries

The rally may require multiple days of travel. Each car can travel four hours (240 minutes) between charges. A car must be plugged into a charger for two minutes for each minute of travel time. Cars start the rally at noon on the first day, fully charged. Cars are permitted remain at a station even after they are fully charged.

It is only possible to drive directly between select pairs of stations. Variations in traffic conditions, road conditions, availability of HOV lanes, etc., result in different travel times along each route depending upon the time of day at which travel along that route begins. All roads are two-way, and the prevailing conditions affect travel in both directions.

The winner is the first car to reach checkpoint ***n***-1, starting form checkpoint 0**.** Other than the starting and ending conditions, cars may pass through the stations in any order, and need not visit all stations to complete the course.

Write a program to determine the earliest time, expressed as the total number of minutes elapsed since the start of the rally, at which a car could reach the final checkpoint.

The Input

There will be several test cases in the input. Each test case starts with a line containing ***n*** ($1 \le n \le 500$), the number of stations, and ***m*** ($1 \le m \le 1{,}000$), the number of connecting road segments.

This is followed by ***m*** blocks, each block describing one road segment. A road segment block has the following structure:

Each block begins with a single line containing two integers, ***a*** and ***b*** ($0 \le a,b \le n-1$, $a \ne b$). These numbers are the two checkpoints connected by that segment. The connections are undirected: a segment permitting travel from station ***a*** to station ***b*** will also allow travel from station ***b*** to station ***a***.

This is followed by from one to twenty 'travel lines' describing travel times. Each of the travel lines contains 3 numbers: ***Start***, ***Stop***, ($0 \le Start < Stop \le 1{,}439$), and ***Time*** ($0 < Time < 1{,}000$). ***Start*** and ***Stop*** are the time of day (expressed in minutes since midnight) described by this line, and ***Time*** is the travel time, in minutes, required to traverse this road segment if travel begins at any time in the range [***Start***..***Stop***], inclusive. The first travel line in a block will have a start time of 0 (midnight, or 00:00). The final travel line in a block will have a stop time of 1439 (i.e., 23:59, or 1 less than 24

hours times 60 minutes). Adjacent travel lines in the input will be arranged in order, and the start time of any line after the first is one higher than the stop time of the preceding line. The travel lines will cover all times from 00:00 to 23:59.

Input will end with a line with two **0**s. All test cases will describe a course that can be completed by the cars.

The Output

For each test case, output a single integer representing the smallest number of minutes needed to complete the rally. Output no spaces, and do not separate answers with blank lines.

Sample Input

```
4 4
0 1
0 1439 100
0 2
0 1439 75
1 3
0 720 150
721 824 100
825 1000 75
1001 1439 150
2 3
0 1439 150
3 2
0 1
0 10 200
11 1439 300
1 2
0 10 200
11 1439 300
4 3
0 1
0 719 500
720 1439 240
1 2
0 964 500
965 1439 2
2 3
0 971 500
972 1439 3
0 0
```

Sample Output

```
180
2360
255
```

Problem E: Escape

You hit the emperor lich with full force and slay it. There is a stair leading upwards here. You climb upstairs. You drink from the pool. You feel much better. The karmic lizard punches through your armor and hits you. You die...

After an epic fight with the emperor lich, the hero struggles to escape the dungeon consisting of n chambers and $n-1$ corridors connecting them. He starts in chamber number 1 and must reach chamber number t, moving only along the corridors. All chambers are reachable from chamber number 1. Bruised after the last fight, the hero starts the journey with 0 hit-points (HP). These points represent his health – if ever they fall below zero, the hero's story ends there as a tragic one.

In some chambers there are monsters – a monster must be fought, and it always manages to take some of the hero's HP. In some other chambers there are magic pools – every pool restores some number of the hit-points. There is no upper limit on the hero's health. Every chamber can be visited multiple times, but the gain or loss of HP happens only once, on the very first visit.

Determine whether the hero can escape the dungeon alive.

Input

The first line of input contains the number of test cases T. The descriptions of the test cases follow:

The first line of each test case contains two integers: the number of chambers n, $2 \leqslant n \leqslant 200\,000$, and the number of the exit chamber t, $2 \leqslant t \leqslant n$. The second line contains n space-separated integers between -10^6 and 10^6 – the i-th of them denotes the HP gain in the i-th chamber (negative denotes a monster, positive – a pool, and zero means that the chamber is empty). The first chamber does not contain a monster, but a pool is possible there. The exit chamber may contain a pool or a monster, and the monster will have to be fought before escaping.

The next $n-1$ lines contain the descriptions of corridors. Each one contains a pair of integers – the ends of a corridor.

Output

For each test case print a single line containing the word **escaped** if escape is possible, or **trapped** otherwise.

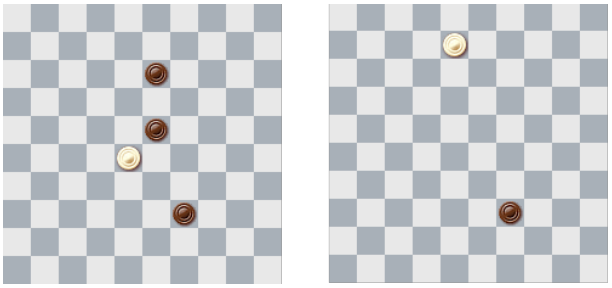
Example

For an example input	the correct answer is:
2 7 7 0 -3 2 2 3 -4 0 1 2 2 3 2 4 1 5 5 6 6 7 3 2 3 3 -4 1 3 2 3	escaped trapped

Problem F: Draughts

Draughts (or *checkers*) is a game played by two opponents, on opposite sides of a 10×10 board. The board squares are painted black and white, as on a classic chessboard. One player controls the dark, and the other the light pieces. The pieces can only occupy the black squares. The players make their moves alternately, each moving one of his own pieces.

The most interesting type of move is *capturing*: if a diagonally adjacent square contains an opponent's piece, it may be captured (and removed from the game) by jumping over it to the unoccupied square immediately beyond it. It is allowed to make several consecutive captures in one move, if they are all made with a single piece. It is also legal to capture by either forward or backward jumps.



The board before and after a single move with two captures.

You are given a draughts position. It is the light player's turn. Compute the maximal possible number of dark pieces he can capture in his next move.

Input

The first line of input contains the number of test cases T. The descriptions of the test cases follow:

Each test case starts with an empty line. The following 10 lines of 10 characters each describe the board squares. The characters # and **.** denote empty black and white squares, **W** denotes a square with a light piece, **B** – a square with a dark piece.

Output

For each test case print a single line containing the maximal possible number of captures. If there is no legal move (for example, there are no light pieces on the board), simply output 0.

Example

For an example input	the correct answer is:
2 .#.#.#.#.# #.#.#.#.#. .#.#.B.#.# #.#.#.#.#. .#.#.B.#.# #.#.W.#.#. .#.#.#.#.# #.#.#.B.#. .#.#.#.#.# #.#.#.#.#. .#.#.#.#.# #.#.#.#.#. .#.#.B.#.# #.B.#.B.#. .#.#.B.#.# #.B.W.#.#. .#.B.B.#.# #.#.#.#.#. .#.B.B.#.# #.#.#.#.#.	2 4

Problem G: History course

You are to give a series of lectures on important historical events, one event per lecture, in some order. Each event lasted for some time interval $[a_i, b_i]$. We say that two events are *related* if their intervals have a common point. It would be convenient to schedule lectures on related events close to each other. Moreover, lectures on unrelated events should be given in the order in which the events have taken place (if an event A preceded an unrelated event B, then the lecture on A should precede the lecture on B).

Find the minimum integer $k \geqslant 0$ and an order of the lectures such that any two related events are scheduled at most k lectures apart from each other (lectures number i and j are considered to be $|i - j|$ lectures apart).

Input

The first line of input contains the number of test cases T. The descriptions of the test cases follow:

The first line of each test case contains the number n, $1 \leqslant n \leqslant 50\,000$. Each of the next n lines contains two integers a_i and b_i, $-10^9 \leqslant a_i \leqslant b_i \leqslant 10^9$ – the ends of the i-th interval. The intervals are pairwise different.

Output

Print the answers to the test cases in the order in which they appear in the input. The first line of the answer to each test case should contain the minimum value of k. The next n lines should list the intervals (in the same format as in the input) in an order such that any two related events are scheduled at most k lectures apart. Remember to put any unrelated intervals in the proper order!

Example

For an example input	a possible correct answer is:
1 3 1 6 2 3 4 5	1 2 3 1 6 4 5

H: Star Simulations

In massive simulations of star systems, we don't want to have to model the gravitational effects of pairs of bodies that are too far away from each other, because that will take excess computing power (and their effects on each other are negligible). We want to only consider the effects two objects have on each other if the Euclidean distance between them is less than ***k***.

Given a list of ***n*** points in space, how many have a distance of less than ***k*** from each other?

Input

There will be several test cases in the input. Each test case will begin with a line with two integers, ***n*** ($2 \le n \le 100{,}000$) and ***k*** ($1 \le k \le 10^9$), where ***n*** is the number of points, and ***k*** is the desired maximum distance. On each of the following ***n*** lines will be three integers ***x***, ***y*** and ***z*** ($-10^9 \le x,y,z \le 10^9$) which are the (***x***,***y***,***z***) coordinates of one point. Within a test case, there will be no duplicate points. Since star systems are generally sparse, it is guaranteed that no more than 100,000 pairs of points will be within ***k*** of each other. The input will end with a line with two **0**s.

Output

For each test case, output a single integer indicating the number of unique pairs of points that are less than ***k*** apart from each other. Output no spaces, and do not separate answers with blank lines.

Sample Input

```
7 2
0 0 0
1 0 0
1 2 0
1 2 3
1000 1000 1000
1001 1001 1000
1001 999 1001
7 3
0 0 0
1 0 0
1 2 0
1 2 3
-1000 1000 -1000
-1001 1001 -1000
-1001 999 -1001
7 4
0 0 0
1 0 0
1 2 0
1 2 3
1000 -1000 1000
1001 -1001 1000
1001 -999 1001
0 0
```

Sample Output

```
3
6
9
```

I - Card Trick

I am learning magic tricks to impress my girlfriend Alice. My latest trick is a probabilistic one, i.e. it does work in most cases, but not in every case. To perform the trick, I first shuffle a set of many playing cards and put them all in one line with faces up on the table. Then Alice secretly selects one of the first ten cards (i.e. she chooses x_0, a secret number between 1 and 10 inclusive) and skips cards repeatedly as follows: after having selected a card at position x_i with a number $c(x_i)$ on its face, she will select the card at position $x_{i+1} = x_i + c(x_i)$. Jack (`J`), Queen (`Q`), and King (`K`) count as 10, Ace (`A`) counts as 11. You may assume that there are at least ten cards on the table.

Alice stops this procedure as soon as there is no card at position $x_i + c(x_i)$. I then perform the same procedure from a randomly selected starting position that may be different from the position selected by Alice. It turns out that often, I end up at the same position. Alice is very impressed by this trick.

However, I am more interested in the underlying math. Given my randomly selected starting position and the card faces of every selected card (including my final one), can you compute the probability that Alice chose a starting position ending up on the same final card? You may assume that her starting position is randomly chosen with uniform probability (between 1 and 10 inclusive). I forgot to note the cards that I skipped, so these cards are unknown. You may assume that the card face of every single of the unknown cards is independent of the other card faces and random with uniform probability out of the possible card faces (i.e. `2-10`, `J`, `Q`, `K`, and `A`).

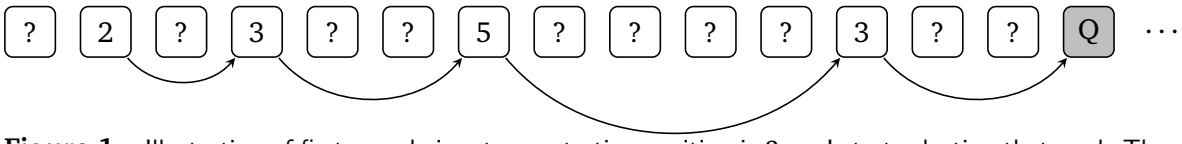


Figure 1 – Illustration of first sample input: my starting position is 2, so I start selecting that card. Then I keep skipping cards depending on the card's face. This process iterates until there are not enough cards to skip (in this sample: `Q`). The final `Q` card is followed by 0 to 9 unknown cards, since `Q` counts as 10.

Input

For each test case:

- A line containing two integers n ($1 \leq n \leq 100$) and m ($1 \leq m \leq 10$) where n is the number of selected cards and m is the 1-based position of my first selected card.
- A line with n tokens that specify the n selected card faces (in order, including the final card). Each card face is given either as an integer x ($2 \leq x \leq 10$) or as a single character (`J`, `Q`, `K`, or `A` as specified above).

Output

For each test case, print one line containing the probability that Alice chooses a starting position that leads to the same final card. Your output should have an absolute error of at most 10^{-7}.

Example

input	output
5 2 2 3 5 3 Q 1 1 A 1 2 A 1 10 A 6 1 2 2 2 2 2 2 7 1 2 2 2 2 2 2 2 3 10 10 J K	0.48713777570233253480715 73 0.10000000000000000000000 0.10000000000000000000000 0.17489233570253142396974 90 0.58307132103217674451174 68 0.62792296111157495562803 50 0.33465658276032720018919 74

Problem J: Captain Obvious and the Rabbit-Man

"It's you, Captain Obvious!" – cried the evil Rabbit-Man – "you came here to foil my evil plans!"

"Yes, it's me." – said Captain Obvious.

"But... how did you know that I would be here, on 625 Sunflower Street?! Did you crack my evil code?"

"I did. Three days ago, you robbed a bank on 5 Sunflower Street, the next day you blew up 25 Sunflower Street, and yesterday you left quite a mess under number 125. These are all powers of 5. And last year you pulled a similar stunt with powers of 13. You seem to have a knack for Fibonacci numbers, Rabbit-Man."

"That's not over! I will learn... **arithmetics!**" *– Rabbit-Man screamed as he was dragged into custody – "You will* **never** *know what to expect... Owww! Not my ears, you morons!"*

"Maybe, but right now you are being arrested." – Captain added proudly.

Unfortunately, Rabbit-Man has now indeed learned some more advanced arithmetics. To understand it, let us define the sequence F_n (being not completely unlike the Fibonacci sequence):

$F_1 = 1$,
$F_2 = 2$,
$F_n = F_{n-1} + F_{n-2}$ for $n \geqslant 3$.

Rabbit-Man has combined all his previous evil ideas into one master plan. On the i-th day, he does a malicious act on the spot number $p(i)$, defined as follows:

$$p(i) = a_1 \cdot F_1^i + a_2 \cdot F_2^i + \ldots + a_k \cdot F_k^i.$$

The number k and the integer coefficients $a_1, \ldots, a_k$ are fixed. Captain Obvious learned k, but does not know the coefficients. Given $p(1), p(2), \ldots, p(k)$, help him to determine $p(k+1)$. To avoid overwhelmingly large numbers, do all the calculations modulo a fixed prime number M. You may assume that $F_1, F_2, \ldots, F_n$ are pairwise distinct modulo M. You may also assume that there always exists a unique solution for the given input.

Input

The first line of input contains the number of test cases T. The descriptions of the test cases follow:

The first line of each test case contains two integers k and M, $1 \leqslant k \leqslant 4000$, $3 \leqslant M \leqslant 10^9$. The second line contains k space-separated integers – the values of $p(1), p(2), \ldots, p(k)$ modulo M.

Output

Print the answers to the test cases in the order in which they appear in the input. For each test case print a single line containing one integer: the value of $p(k+1)$ modulo M.

Example

For an example input	the correct answer is:
2 4 619 5 25 125 6 3 101 5 11 29	30 83

Explanation: the first sequence is simply $5^i \mod 619$, therefore the next element is $5^5 \mod 619 = 30$. The second sequence is $2 \cdot 1^i + 3^i \mod 101$.

Problem K: Digraphs

A *digraph* is a graph with orientation... oh, sorry, not this time. Let's stop being nerds for a minute and talk about languages (*human* languages, not PHP).

Digraphs are pairs of characters that represent one phoneme (sound). For example, "ch" in English (as in "church") is a single consonant sound. The languages of Central Europe are fond of digraphs: Hungarian "sz", Czech "ch" and Polish "rz" are fine examples of them.

Digraphs are very annoying for people who do not use them natively. We will make up a letter-puzzle specifically for those people. Given a list of digraphs, construct a biggest possible square of lower case English letters such that its rows and columns *do not* contain any of these digraphs. This means that no two consecutive letters (read from top to bottom or from left to right) can form a digraph.

Input

The first line of input contains the number of test cases T. The descriptions of the test cases follow:

Each test case starts with an integer n, $0 \leqslant n \leqslant 676$, denoting the number of forbidden digraphs. The n following lines contain the digraphs.

Output

For each test case print a square of the largest possible size which does not contain any of the digraphs. If it is possible to construct a square of size 20×20 or bigger, print only a 20×20 square.

Example

Part of the example test data below was omitted for clarity. You can access full sample tests at your workstation.

For an example input	a possible correct answer is:
2 628 aa az ba bb bc ... by ca cb cc ... cy da ... dy ... wa ... wy ya ... yy za zb ... zy zz 2 aa bb	aw wz abababababababababab babababababababababa abababababababababab babababababababababa abababababababababab babababababababababa abababababababababab babababababababababa abababababababababab babababababababababa abababababababababab babababababababababa abababababababababab babababababababababa abababababababababab babababababababababa abababababababababab babababababababababa

Problem L: Bus

A bus with n passengers opens its door at the bus stop. Exactly half of its passengers and an additional half of a passenger get out. On the next stop, again, half of the passengers plus half of a passenger leave the bus. This goes on for k stops in total. Knowing that the bus leaves the last stop empty, and that no one was hurt during the trip, determine the initial number n of people in the bus.

Input

The first line of input contains the number of test cases T. The descriptions of the test cases follow:

The only line of each test case contains the number of stops k, $1 \leqslant k \leqslant 30$.

Output

For each test case, output a single line containing a single integer – the initial number of bus passengers.

Example

For an example input	the correct answer is:
2 1 3	1 7