Problem A
Mystic Craft

In the Ancient Clash of Mystic Pandas (ACM Pandas) game, the player plays the role of a Panda Knight who needs to defend Panda Land by defeating evil Panda Wizards. As the wizards have special magical powers, they need to be defeated using specific types of Mystic Sticks (regular bamboo sticks are ineffective against them). To obtain a Mystic Stick, Panda Knight needs to craft it by infusing his regular bamboo stick with several different kinds of magic shards according to the known recipe for that particular type of Mystic Stick.

Normally, all kinds of shards can be purchased for 1 gold/piece from the Panda Magic Store (therefore Panda Knight will have no problem of acquiring the shards, as long as he has enough gold to buy them all from the Store). However due to the recent invasion, to conceal the shards from the incoming Panda Wizards, Panda Magic Store has packaged all the shards into inconspicuous Mystery Boxes. A Mystery Box contains a random piece of magic shard which type can't be determined prior to buying and opening the box, which Panda Knight can also buy for the same price (1 gold/box).

As his Panda Knight character is not rich, Mr. Wah is concerned about the possibility that he's unable to buy all the necessary shards due to not getting the required amount of a specific type of shard. He needs your help! Your task as Mr. Wah's best programmer friend is to compute the probability that he will be able to get all the shards and craft the Mystic Stick, so that he can plan his playing strategy accordingly. You can safely assume that all the required shards are available on the store via the Mystery Boxes, that the boxes will only contain the needed types of shard, and that each type of shard has equal probability of being contained inside any particular Mystery Box.

Input

The first line of input contains an integer T ($1 \le T \le 100$), the number of test cases follow. Each test case begins with two integers G and N ($1 \le N \le G \le 32$) in one line, denoting the amount of Panda Knight's gold and the number of needed magic shard types respectively. The next line contains N integers, denoting how many magic shards of each type ($1 \le M_1 \ldots M_N \le 32$, $M_1 + \ldots + M_N \le G$) are needed to craft the Mystic Stick.

Output

For each of the test cases, print the test case number followed by the probability (in percentage, correct to 6 decimal places – Mr. Wah is paranoid about this game) that Panda Knight will be able to craft the Mystic Stick, with the format as shown by the sample output.

Sample Input

```
5
3 2
1 1
8 3
3 2 2
10 3
1 2 3
7 7
1 1 1 1 1 1 1
32 8
1 1 1 1 1 1 1 1
```

Output for Sample Input

```
Case #1: 75.000000
Case #2: 23.472032
Case #3: 58.934106
Case #4: 0.611990
Case #5: 89.127753
```

Explanation for 1st sample test case:

There are 2^3 = 8 possible combinations of shard types that Knight Panda can get by purchasing 3 Mystery Boxes with his available gold:

1. Type 1, Type 1, Type 1
2. Type 1, Type 1, Type 2
3. Type 1, Type 2, Type 1
4. Type 1, Type 2, Type 2
5. Type 2, Type 1, Type 1
6. Type 2, Type 1, Type 2
7. Type 2, Type 2, Type 1
8. Type 2, Type 2, Type 2

Since there are 6 out of 8 combinations that gives required amount of shards (at least one Type 1 shard and one Type 2 shard), the probability that Panda Knight will be able to craft the Mystic Stick is 6/8 = 75%.

Problem B

Top 10

We all use search everyday; to find a file in a directory, to find an email in the inbox, to find a song in a playlist. Search is more than just a linear scan through a list of texts in a dictionary; It's binary search, it's indexing, it's using your full text search algorithms to solve one of the hardest problems we know.

-- adapted from NUMB3RS

Given a dictionary containing less than N = 20000 words labeled from 1 to N. Each word consists of lowercase characters (from 'a' to 'z') with arbitrary length. The total number of characters in the dictionary is at most 100,000. Your task is to answer at most Q = 100000 queries. Each query q_i is also a word (as defined above). For each query, you have to print the "Top 10" words in the dictionary with the following rules:

- All the words in the "Top 10" have to contain the substring q_i.
- All the words in the "Top 10" have to be sorted in this order:
  1. The words with shorter length come first, if they have equal length then
  2. The lexicographically smaller words come first, otherwise
  3. The words with smaller label come first.
- If the number of words in the dictionary that contains the substring q_i is less than 10 then print all the words otherwise, print only the top-10 words (note: the words are printed using their labels).
- If there is no word in the dictionary that contains the substring q_i then print "-1" (without the quotes).

Input

The first line contains the number N. The next N lines contains the N words in the dictionary (the ith line is the word with label i). The next line contains the number Q followed by the Q lines containing the queries.

Output

For each query, print one line containing the labels of the "Top 10" words (separated by a space) in the dictionary using the rules defined above.

Sample Input

```
17
acm
icpc
regional
asia
jakarta
two
thousand
and
nine
arranged
by
universitas
bina
nusantara
especially
for
you
5
a
an
win
b
z
```

Output for Sample Input

```
1 8 4 13 5 10 3 7 14 15
8 10 7 14
-1
11 13
-1
```

Problem C
Random Simple Polygon

Given N ($3 \le N \le 10{,}000$) non-collinear triplewise points on a 2-D plane, select K ($3 \le K \le N$) points and order them such that the ordered points form a simple K-sided polygon. A polygon is called simple if no pair of its sides cross each other.

Input

First line of the input contains a positive integer T ($1 \le T \le 10$), the number of cases. Follows afterwards are the description of each case. For each case, the first line contains two integers N and K. Each of N following lines contains two integers, x_i, y_i ($1 \le x_i$, $y_i \le 1{,}000{,}000$), one of the given points. For simplicity, all points are numbered from 1 to N according to the order of appearance in the input. There is no blank line separating each case.

Output

For each case, output K integers, each on a line, the selected points in the order how the K-sided simple polygon is constructed. If there are more than one possible ordered selection that fits the criteria, output any one. It is always possible to form at least one simple K-sided polygon. There is no blank line separating each case.

Sample Input	Output for Sample Input

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

Problem D
Alien

It is 2050 and human live with alien. There are two kind of alien: good alien and evil alien. Good alien were our friend because they help us to develop good technology for our earth. Evil alien were very very bad, they develop heavy and dangerous weapons that can destroy earth. Alien were very hard to kill because of their ultra regeneration skill. Luckily, we have developed a kind of bomb that can kill alien in an instant. This weapon is very expensive so we must use it wisely.

Given a map of 10 x 10, each element in the map will be:

- '`.`' represents empty region.
- '`g`' represents good alien.
- '`e`' represents evil alien.

Calculate how many evil alien that can be killed without killing any good alien, and how many bombs we need to do that. A bomb has a 3x3 area of effect, so all of the 8 adjacent neighbors will also be destroyed. You can put bomb anywhere in the map, even if there is an alien in that cell.
For example,

```
e..
.Xe
..e
```

Bombing at `X` will kill three evil aliens.

Our spy has reported that the number of alien groups from each side is less than 16.

Input

First line in the input will be *T* ($1 \le T \le 100$) number of cases. Each case will have a map of 10 x 10 represent the city. Between each city will be separated by a blank line.

Output

For each case, output two numbers: *a* and ***b*,** where *a* is the maximum number of evil aliens that you can kill and *b* is the minimum number of bombs you need to do that.

Sample Input	Output for Sample Input
`2`	`13 2`
`..........`	`2 2`
`..........`	
`...eee....`	
`...eee....`	
`...eee....`	
`..........`	
`..........`	
`...ee.....`	
`...ee.....`	
`..........`	
`..........`	
`..........`	
`...ggg....`	
`...geg....`	
`...ggg....`	
`..........`	
`..........`	
`...eg.....`	
`...ge.....`	
`..........`	

Problem E

A + B

Given two integers *A* and *B* that are not necessarily in base-10, find the smallest possible *A* + *B* in base-10.

For example,

A = 213	possibly base-4 (39 in base-10)
B = 4721	possibly base-8 (2513 in base-10)

A + *B* = 39 + 2513 = 2552

Input

First line of the input contains a positive integer *T* (1 ≤ *T* ≤ 100), the number of cases. Each case contains two positive integers *A* and *B*. *A* and *B* will contain at most 5 digits, each digit will be between 0 and 9, inclusive, and no leading zeroes.

Output

For each case, output an integer in base-10 denoting the smallest possible *A* + *B*.

Sample Input	Output for Sample Input
`3`	`2552`
`213 4721`	`99`
`1001 90`	`592`
`638 241`	

Problem F

1:1000000000000...

Do you know Lotto? Lotto is a game of probability. Six numbers are drawn from a range of numbers (such as 42, 47, 47, 49, 51, and 54). Michigan, for instance, has a 6-out-of-47 game (6/47), meaning that six numbers are drawn from possible 47. Florida's Lotto is 6/53, meaning that six numbers are drawn from a possible 53.

To play Lotto, indicate your six chosen numbers by marking the numbered squares on a play slip. Then take the play slip to a lottery retailer (or agent). The retailer enters your selection in the on-line terminal, which produces your game ticket. The ticket, not the play slip, is the official receipt and must be presented and validated in the event of a win. Always check to make sure that the correct date and numbers are on the game ticket before you leave. Lottery agents are found in convenience stores, gas stations, and grocery stores.

The cost for one chance at Lotto is still $1 in many states. So for one chance, or play, at Lotto, you would pay $1. For five plays -- that is, to play five sets of numbers--you would pay $5. Illinois offers a bargain: two plays for $1.

Typically, Lotto drawings are held twice a week, usually on Wednesday and Saturday nights. However, this may not be true for every state.

The lottery officials use special ball-drawing machines, and the balls are numbered. The machine randomly shoots out six selected balls; these balls display the winning numbers for that evening's lottery drawing. If all six of your numbers exactly match the numbers drawn, you win the jackpot. In Lotto, your numbers don't need to be listed in any particular order, as long as they match those drawn. (taken from http://entertainment.howstuffworks.com/how-to-play-the-lottery1.htm)

In most case of lottery they don't draw same number, but in our problem here same number are allowed. So here, you can play number "42 42 45 45 45" which is not allowed in common lottos.

They say there's more chance of you getting hit by a car on your way to buy a lotto ticket than the chance of winning one. Is the probability to win really that small? You are about to help me find out.

Input

Input starts with an integer *T* (1 ≤ *T* ≤ 10), the number of test cases. *T* input blocks follows. Each input blocks starts with an integer *M* (1 ≤ *M* ≤ 50), meaning that the game involves *M* distinct numbers particularly from 1 to *M*. You don't know how the ball-drawing machine works, but you do know that for there is a constant probability for each number to pop up. The second line of each input blocks will consists of *M* real-numbers P_1, P_2, P_3, P_4, ..., P_M, where P_i denotes the probability of ball with number *i* to pop-up. You can safely assume that the total probability is equal to 1.

The third line of input is an integer *Q* (1 ≤ *Q* ≤ 100), the number of queries. *Q* lines follow with each line begins with an integer *N* (1 ≤ *N* ≤ *M*). So if N is 6, you are playing 6-out-of-M game in this particular query. *N* numbers follows, those are your chosen number. You are to determine what is the probability of that number winning the lottery. Since the probabilities can be very low, output them in scientific notation: `a x 10^b` where `a` should be between 1.00000 and 9.99999, inclusive. Output `a` in 5 decimal places.

Output

For each query, you are to determine the probability of that number winning the lottery.

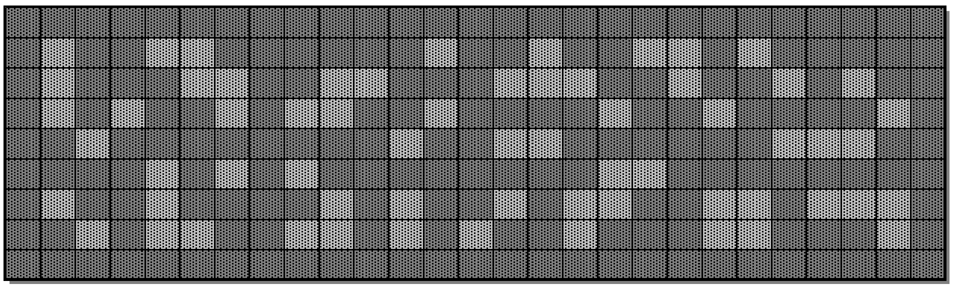
Sample Input	**Output for Sample Input**

```
2
5
0.2 0.2 0.2 0.2 0.2
3
1 2 3
5 5 5 5
3
3
0.2 0.3 0.5
2
1
2 1
```
	```
	Test Case #1:
	4.80000 x 10^-2
	1.60000 x 10^-3
	2.00000 x 10^-1
	Test Case #2:
	2.00000 x 10^-1
	1.20000 x 10^-1
	```

Problem G

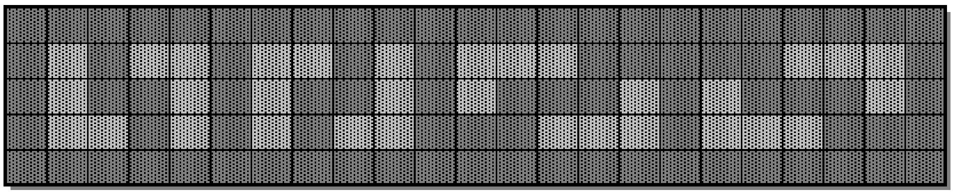
The Puzzle Board from God

Once upon a time, there was a man known as Mr. Bonus (Born at Binus), no one know his real name. One day, after miles of walk, he sat and took a rest. Accidentally, a board fell from sky and hit his head. He was very angry because he thought someone had intentionally thrown that board, but he did not see anyone around. So he thought that board is from God. He tried to look at that board and found a lot of puzzle shape like this:

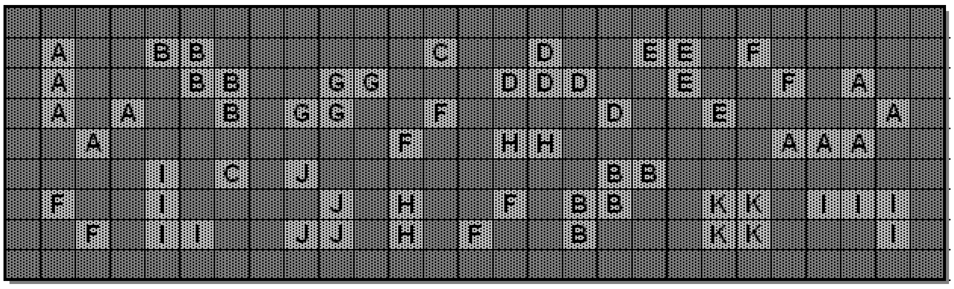


He thought that God want him to calculate how many different shapes are there in that board. Mr. Bonus: “This is very easy, I just need to open my BinusBerry to make a program to calculate it.” But he thought that he should take a rest, so he called you to solve his problem.

You are about to count how many different shapes are there in the board. Two shapes are considered the same if they can be matched by rotating (90, 180, or 270 degree) and/or mirroring. These 8 shapes are considered as the same shape:



You should output the number of different shapes and label the puzzles. The same puzzle should use the same label. This is the result that Mr. Bonus wanted:



Input

Input consists of several cases. Each case begins with two integer *R* and *C* (1 ≤ *R*, *C* ≤ 100) the number of rows and columns of the board. Next, *R* rows follow each with *C* characters (either '0' or '1') which correspond to the puzzle. Input will be ended by R = C = 0.

Output

For each case, first print in a line the number of different shapes. Next, print the board (*R* lines) and replace each shape with label (A-Za-z), in such a way that, when the rows of the board are concatenated from top to bottom, the resulting string is lexicographically smallest. Use the same label for the same shape. There will be at most 52 different shapes and 500 shapes. Put a blank line between cases.

Sample Input

```
9 27
000000000000000000000000000
010011000000100100110100000
010001100110001110010010100
010100101100100001001000010
001000000001001100000011100
000010101000000001100000000
010010000101001011001101110
001011001101010010001100010
000000000000000000000000000
0 0
```

Output for Sample Input

```
11
000000000000000000000000000
0A00BB000000C00D00EE0F00000
0A000BB00GG000DDD00E00F0A00
0A0A00B0GG00F0000D00E0000A0
00A00000000F00HH000000AAA00
0000I0C0J00000000BB00000000
0F00I0000J0H00F0BB00KK0III0
00F0II00JJ0H0F00B000KK000I0
000000000000000000000000000
```

Problem H

Hero’s Adventure

There is a game production company that is going to develop a new adventure game. You’re very interested and join the recruitment phase. You’re being tested to solve this simulation:

- For each simulation there will be an *R* x *C* map with the following characters:
  - '#' represents wall, hero can't move here.
  - 'S' represents starting point of the hero.
  - 'F' represents finish point, the game ends.
  - Number '1' to '9' represents the life point to be reduced if the hero moves here.
  - 'E' represents enemy that the hero need to fight if the hero want to move here.
  - 'I' represents item that will add 100 points to the hero's life force, become '0' after taken (no life force will be reduce to come back to this point).
- The hero has life force (*Lf*), Experience (*Exp*), Defend power (*Def*) and Strength (*Str*).
- The hero will be given extra 1 (one) point for the strength and defend power for each 25 (twenty five) experience point gained by the hero.
- Hero will die if *Lf* ≤ 0.
- Rules of the fight with enemy:
  - Each enemy will have Strength, Life force and Experience.
  - The hero and the enemy will be given one chance to attack the other each turn start with the hero, until either the hero or the enemy die (life force ≤ 0).
  - If the hero attacks the enemy, the enemy's life force will be decreased by the hero's strength.
  - If the enemy attacks the hero, the hero's life force will be decreased by the enemy's strength reduced by the hero's defend power.
  - The hero will be given the enemy's experience if the hero defeats the enemy.
  - After the enemy defeated, the place will become '0' (no life force will be reduce to come back to this point).

For each simulation, you’re asked to print three paths, *Lf* and *Exp* to the finish point that give:

1. The highest *Lf*.
2. The highest *Exp.* If there’s more than one highest *Exp*, choose the highest *Lf*.
3. The shortest way. If there’s more than one shortest way, choose the highest *Lf*.

If there is multiple path, choose the lexicographically smallest one.

Input

Input consists of several cases. Each case begins with four integers in a line: *Str* (3 ≤ *Str* ≤ 10), *Def* (0 ≤ *Def* ≤ 5), *Exp* (0 ≤ *Exp* ≤ 20) and *Lf* (10 ≤ *Lf* ≤ 1000). Denoting the strength, defend, experience and life force of the hero respectively. The next line contains *C* (3 ≤ *C* ≤ 20) and *R* (3 ≤ *R* ≤ 20) denoting the columns and rows of the map respectively. The next *R* lines contain the map as described above. The next lines contain the description of the enemy, *eStr* (0 ≤ *eStr* ≤10), *eLf* (10 ≤ *eLf* ≤ 100) and *eExp* (1 ≤ *eExp* ≤ 25) denoting the enemy’s strength, life force and experience to be gained if it dies respectively. Each enemy data corresponds to each ‘e’ which appear first in the map.

Output

For each simulation print "Simulation #n" without double quote(") and #n replace by the simulation number start with 1. The next three lines each contain path, life force and experience. Use ‘L’ (left), ‘R’ (right), ‘D’ (down), ‘U’ (up) to describe the path that should be taken by the hero.

If it’s not possible to reach the finish point then print "No solution." without double quote (“) instead of the three lines. Print blank line between simulation.

Sample Input	Output for Sample Input

`10 5 20 1000`
`3 3`
`###`
`SEF`
`###`
`10 50 15`
 	
`10 0 0 100`
`7 4`
`#######`
`S12E#2#`
`#IE456F`
`#######`
`3 10 5`
`10 100 20`
 	
`10 0 0 10`
`3 3`
`###`
`SEF`
`###`
`10 50 15`
	`Simulation 1`
	`RR 980 35`
	`RR 980 35`
	`RR 980 35`
	
	`Simulation 2`
	`RDURRDRRR 181 5`
	`RDRRUDRRR 90 25`
	`RDRRRRR 94 20`
	
	`Simulation 3`
	`No solution.`

Problem I
Spam Detection

It is well-known that the number of occurrences of the term "free" can distinguish spam and non-spam emails. Your task is to build a spam detection module, based on the number of term "free" in an email.

The core of this detection module is a spam classifier, which is represented by two variables: *Low* and *High*. An email that contains *X* "free" words is classified by this module as a spam if $Low \le X \le High$, otherwise it is not.

To measure the goodness of a classifier, we introduce several information-retrieval terminologies:

		Actual	
		Spam	Non-Spam
Predicted	Spam	TP	FP
	Non-Spam	FN	TN

TP (true positive) is the number of emails which are truly predicted as spam; FN (false negative) is the number of emails which are wrongly predicted as non-spam, and so on.

The portion of emails that are correctly identified as spam is denoted as precision (P), which is formulated as P = TP / (TP + FP). The portion of spam emails that are successfully identified is denoted as recall (R), which is formulated as R = TP / (TP + FN). To balance between precision and recall, we use the F-measure which is formulated as F = 2 x P x R / (P + R). For example, when TP = 5, FP = 3, FN = 2, TN = 4, we have R = 5/7, P = 5/8, and F = 2/3.

When there is no spam, we can report all emails as non-spam with F = 1.0 (perfect classifier).

Our data mining team has manually analyzed several emails and labeled them as spam or non-spam. Your task is to find the values of *Low* and *High* that yield the best classifier, i.e., the one that maximizes the F-measure.

Input

The input consists of several test cases, where each case contains of two lines:

N	: The maximum number of term “free” in any emails $(1 \le N \le 2 \text{ x}10^6)$
a_0 A B M	: parameters of random number generator. $(2 \le M \le 10; 0 \le a_0, A, B < M)$

This random number generator generates a sequence of number:
$a_i = (A * a_{i-1} + B) \text{ MOD } M$ for i >= 1

Specifying:
$pos_i = a_{2i}$ $(0 \le i \le N)$: the number of spam emails with i number of term “free”.

$neg_i = a_{2i+1}$ $(0 \le i \le N)$: the number of non-spam emails with i number of term “free”.

The input is terminated by EOF.

Output

For each simulation print the F-measure of the best classifier (accurate to 6 decimal places).

Sample Input	Output for Sample Input

3
1 1 1 3
5
2 3 4 5
	0.666667
	0.923077

Explanation for the 1st case:

This random number generator generates a sequence of 1, 2, 0, 1, 2, … The number of spam emails is: pos_i = {1, 0, 2, 1}, and the number of non-spam emails is neg_i = {2, 1, 0, 2}.

The optimal classifier treats emails with number of term “free” between 2 and 3 as spam, with R = 3/4 and P = 3/5, resulting F = 2/3. Another way of producing optimal classifier is to consider emails with number of term “free” equals to 2 as spam.

Problem J

Favorite Time

Freddy has just built his own homepage and he was very happy. Now he wants to add a new feature for his homepage, favorite time detector. With this feature, he can find out when most users visit his homepage. He has successfully retrieved visiting time for each online user, but he didn't know how to find the time range when most users visit his page.

Given *N* range of time when users visit his page, you're going to help Freddy to find the time range when most users visit his page.

Input

Input begins with an integer *T* (1 ≤ *T* ≤ 100) denoting the number of cases. Each case begins with an integer *N* (1 ≤ *N* ≤ 100) denoting the number of user visiting Freddy's page. The next *N* lines each will contains the hour and minute when each user visit Freddy's page, ah:ai bh:bi (00 ≤ ah, bh ≤ 23; 00 ≤ ai, bi ≤ 59) denoting the beginning and end time, respectively. ah:ai will always be equal to or smaller than bh:bi. ah, ai, bh and bi will always be two digits (leading zero when it is necessary).

Output

For each case, print the time range when most users visit his page in sh:si th:ti format where sh:si is the beginning time and th:ti is the end time. sh, si, th, ti should be two digits (leading zero when it's necessary). If there are two or more time ranges which have the same number of visiting users, output the first occurrence.

Sample Input	Output for Sample Input
2 3 00:00 10:25 13:07 15:09 09:00 14:00 3 19:00 19:30 19:31 20:00 19:15 19:45	09:00 10:25 19:15 19:45