IEEEXtreme 8.0 October 18, 2014

List of all problems

Binary matrices

Help IBM puzzlemaster to test answers to January's 2014 challenge (http://domino.research.ibm.com/Comm/wwwr_ponder.nsf/Challenges/January2014.html).

Input

First line: **N, M** , the size of the matrix (1<=N<100 and 1<=M<=10).
Next **N** lines: **M** bits in each line.
Note: Please pay attention to the different limits (for N and M) used in this challenge as opposed to the IBM challenge and to the fact that in this challenge the input consists of N * M integers while the sample matrix at IBM website has N strings with M digits.

Output

In the first line of the output, the number of errors **K** should be printed.
Then **K** lines should follow, listing all errors.
At first all type 1 errors should be reported in the format:
i1=row_index
then type 2 errors should be reported in the format:
i1=row_index_1 i2=row_index_2
sorted in lexicographical order.
(please note the single space character between the row_index_1 and i2)

If there are no errors, the program should output 0.

Note 1: The row indices are assumed to start from value 1.
Note 2: There is a newline character at the end of the last line of the output.

Sample Input 1

```
3 2
0 0
0 1
1 1
```

Sample Output 1

```
0
```

Sample Input 2

```
5 3
0 0 0
0 1 0
1 1 1
0 1 1
0 0 1
```

Sample Output 2

```
2
i1=1
i1=1 i2=4
```

Star Fleet Express

A research team at the Star Fleet Academy is close to perfecting the first warp drive. They are planning to create a commercial space flight company, Xtreme Space Lines (XSL).

Task

The research team would like you to write a trip planner program that finds the fastest trip on XSL, if one exists, given a passengers desired start and end location and their earliest departure time. If the shortest trip involves one or more transfers, each stopover must be at least 1 hour long.

Input

The first line in the test case contains three integers: the number of space ports, *p* , 1 <= *p* <= 1,000; the number of planned flights, *f*, 1 <= *f* <= 500,000; and the number of passenger queries, *q* , 1 <= *q* <= 100.
On the next *p* lines are names of space ports, one per line, which are comprised only of alphanumeric characters and the underscore.
Next are *f* lines, each describing a flight, in the form:

[Start] [Start_Date] [Start_Time] [End] [End_Date] [End_Time]

Where

[Start] is the departure location for a flight
[Start_Date] represents date in UTC when the flight leaves
[Start_Time] represents time in UTC when the flight leaves
[End] is the arrival location for a flight , which must be different than> + [Start]
[End_Date] represents date in UTC when the flight arrives
[End_Time] represents time in UTC when the flight arrives

Next are *q* lines, each describing a query, in the form:

[Start] [Start_Date] [Start_Time] [End]
Where
[Start] is the desired departure location for a passenger
[Start_Date] represents earliest start date in UTC for a passenger

[Start_Time] represents earliest start time in UTC for a passenger
[End] is desired arrival location, which must be different than> + [Start]

The dates are non-negative integers,each less or equal to than 2 billion, equal to the number of days after October 18, 2014. The times are non-negative integers equal to the number of minutes after midnight, i.e. they are integers between 0 and 1439, inclusive. The locations provided in the flights and the queries are all drawn from the list of *p* space ports provided in the test case.

Output

Your program should output the date and the time, in the same form as the input, of the earliest possible arrival for each query. If there is no trip possible, your program should output "No trip on XSL".
Note: Every line of output should end in a newline character .

Sample Input 1

```
3 5 4
Earth
Romulus
Vulcan
Earth 1 720 Romulus 49 120
Earth 200 1080 Vulcan 786 0
Earth 200 1000 Romulus 1544 1439
Romulus 50 720 Earth 149 240
Vulcan 786 60 Romulus 1543 1100
Earth 0 680 Romulus
Earth 2 800 Romulus
Vulcan 800 800 Romulus
Romulus 2 900 Vulcan
```

Sample Output 1

```
49 120
1543 1100
No trip on XSL
786 0
```

Sample Input 2

```
5 4 3
Earth
Romulus
Vulcan
Kronos
Ceti_Alpha_V
Earth 1 720 Vulcan 49 120
Vulcan 49 179 Romulus 824 0
Romulus 825 720 Earth 1654 0
Earth 700 600 Vulcan 749 1420
Earth 0 680 Romulus
Earth 830 800 Vulcan
Kronos 800 800 Ceti_Alpha_V
```

Sample Output 2

```
No trip on XSL
No trip on XSL
No trip on XSL
```

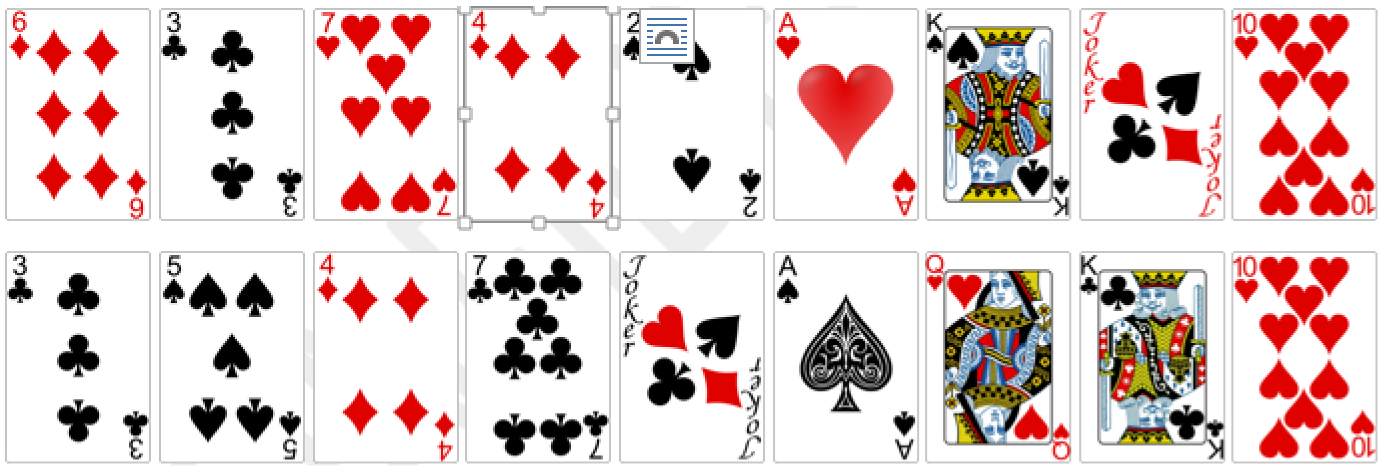
Kabloom

The card game *Kabloom* is played with multiple decks of playing cards. Players are dealt 2 *n* cards, face up and arranged in two rows of *n* cards. The players must discard some of the cards, so that the cards that remain in the first row match the rank of the cards that remain in the second row. The cards match only in rank (e.g. an *Ace* of *Hearts* matches any other *Ace* regardless of suit), but they must appear in the same order in each row. The players are not able to rearrange the order in which the cards appear. Note also that a *Joker* can match any card including another *Joker* .

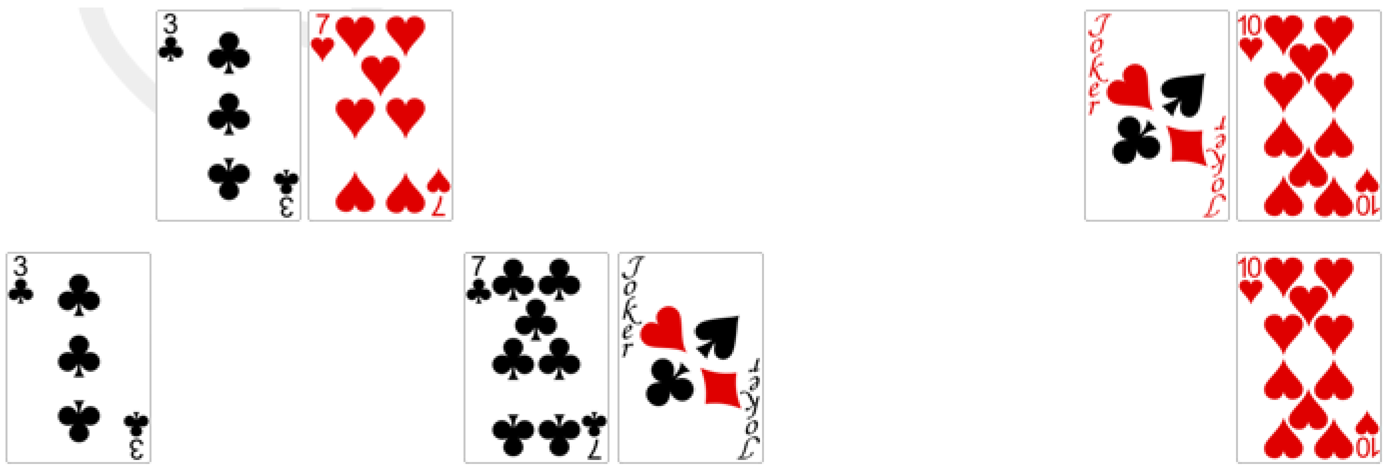
The goal is to maximize the sum of the point value of the cards that remain. *Aces* are worth 20 points, face cards are worth 15 points, and the numbered cards are worth the number on the card (e.g. the *Seven* of *Clubs* is worth 7 points).The value of a *Joker* is equal to the card with which it is matched, e.g. a *Joker* matched with an *Ace* is worth 20 points, a *Joker* matched with a face card is worth 15 points, etc. If two *Jokers* are matched with each other, they are worth 50 points each.

Task

Write a program that determines the value of the best hand given the two rows of cards. For example, consider the hand that is dealt below.



The best possible hand has a value of 140, and is obtained by keeping the cards shown below:


Card Images by Byron Knoll

Input

The input is made up of multiple test cases (#test cases<=30, if 1<=n<=10 or #test cases<=10 if 10<=n<=1000). Each test case contains three lines of input.
The first line in each test case is an integer *n* , 1 <= *n* <= 1,000, indicating how many cards are in each row.
The second line of the test case will contain *n* symbols representing the ranks of the cards in the first row. Each symbol will be chosen from the list {A, 2, 3, 4, 5, 6, 7, 8, 9, T, J, Q, K, R}. The symbols in the list represent the following ranks, respectively, {Ace, Two, Three, Four, Five, Six, Seven, Eight, Nine, Ten, Jack, Queen, King, Joker}. Similarly, the third line of the test case will contain the *n* symbols of the cards in the second row.
The input will end with a 0 on a line by itself.

Output

For each test case, output the value of the best Kabloom hand on a line by itself. Note that the cards that comprise the best Kabloom hand may not be unique for a test case.
Note: Every line of output should end in a newline character .

Sample Input 1

```
9
6 3 7 4 2 A K R T
```

```
3 5 4 7 R A Q K T
0
```

Sample Output 1

```
140
```

Sample Input 2

```
7
R R 5 4 A T Q
Q 3 T A 8 8 8
13
A 2 3 4 5 6 7 8 9 T J Q K
K Q J T 9 8 7 6 5 4 3 2 A
6
A A A A A A
K Q J T 9 8
13
A 2 3 4 5 6 7 8 9 T J Q K
A 2 3 4 5 6 7 8 9 T J Q K
0
```

Sample Output 2

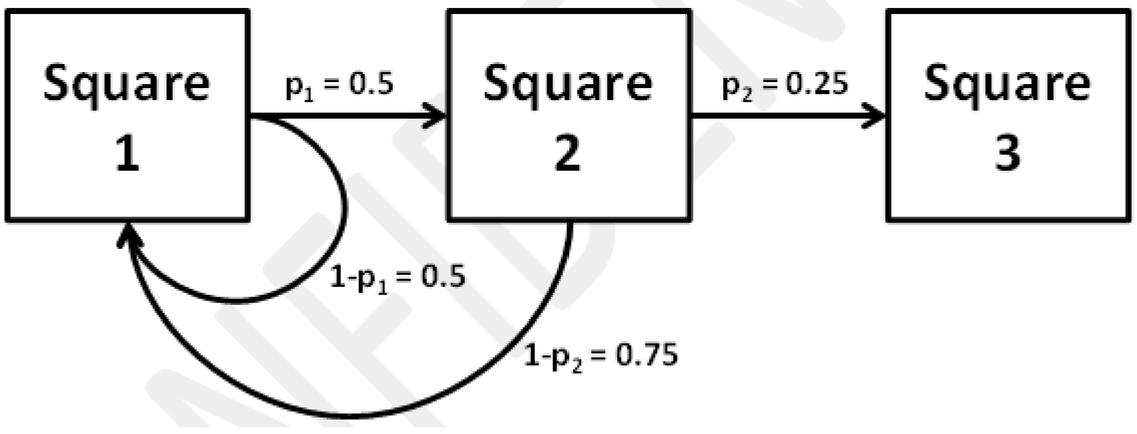
```
90
40
0
238
```

Back to Square 1

The game “Back to Square 1” is played on a board that has *n* squares in a row and *n-1* probabilities. Players take turns playing. On their first turn, a player advances to square 1.After the first turn, if a player is on square *i* , the player advances to square *i* + 1 with probability *p(i)* , and returns to square 1 with probability 1-*p(i)* .The player is finished upon reaching square *n* .

Task

Write a program that determines the expected number of turns needed for a player to reach the final square.For example, consider the board below with *n* = 3 and *p(1)* = 0.5 and *p(2)* = 0.25. A player moves to square 1 on their first turn. With probability *p(1)* , they move to square 2 on their second turn, but with probability 1- *p(1)* , they remain on square 1. If they were lucky and made it to square 2 on their second turn, they advance to square 3 on their third turn with probability *p(2)* , but they would go back to square 1 with probability 1- *p(2)* . Thus, a really lucky player could finish is 3 turns. However, on average, it would take 13 turns for a player to make it to square 3.



Input

The input is made up of multiple test cases. Each test case contains 2 lines of input.
The first line in each test case is an integer *n* , 1 <= *n* <= 1,000, which represents the number of squares for this test case.
On the next line are *n* -1 single-space separated floating point numbers, each greater than 0 and less than or equal to 1, representing *p(1)* , *p(2)* , *p(3)* , ..., *p(n-1)* , respectively.
The input will end with a 0 on a line by itself.

Note: If for an input test case n=1 (i.e. there is only one square) then there will be no following line since there will be no probabilities. For example, the following input:
2
0.5
1
3
0.1 0.2
0
contains in total 3 test cases. The first one having 2 squares with an in-between transition probability equal to 0.5, the second test case consists of a single square (and thus no transition probabilities are provided) and the last test case consists of 3 squares with respective transition probabilities equal to 0.1 and 0.2 .

Output

For each test case, output the expected number of turns needed to reach the final state, **rounded to the nearest integer**. You are guaranteed that the expected number of turns will be less than or equal to 1,000,000.
Note: Every line of output should end in a newline character .

Sample Input 1

```
3
0.5 0.25
0
```

Sample Output 1

```
13
```

Sample Input 2

```
2
0.5
4
```

```
0.3 0.2 0.1
0
```

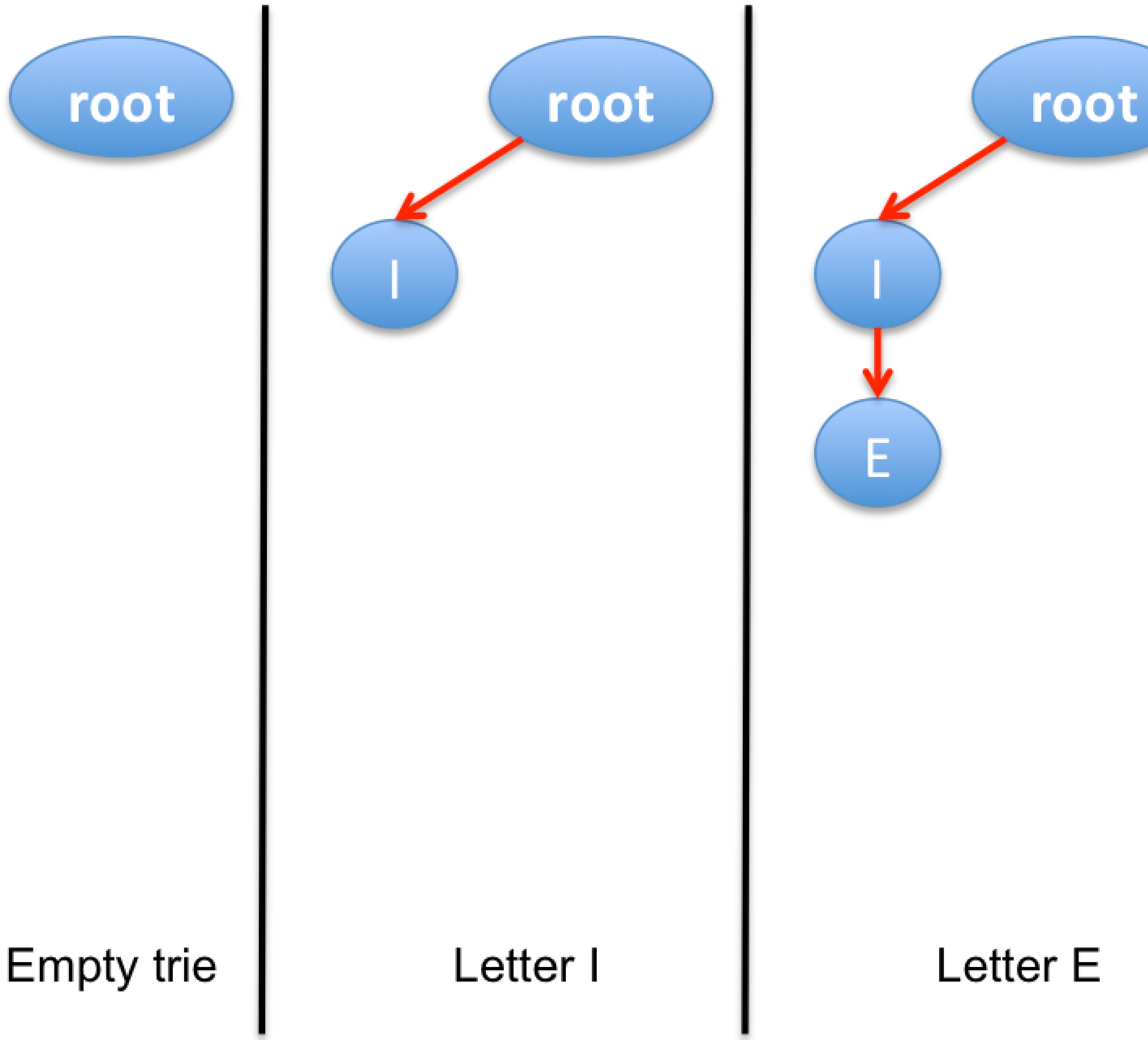
Sample Output 2

```
3
228
```

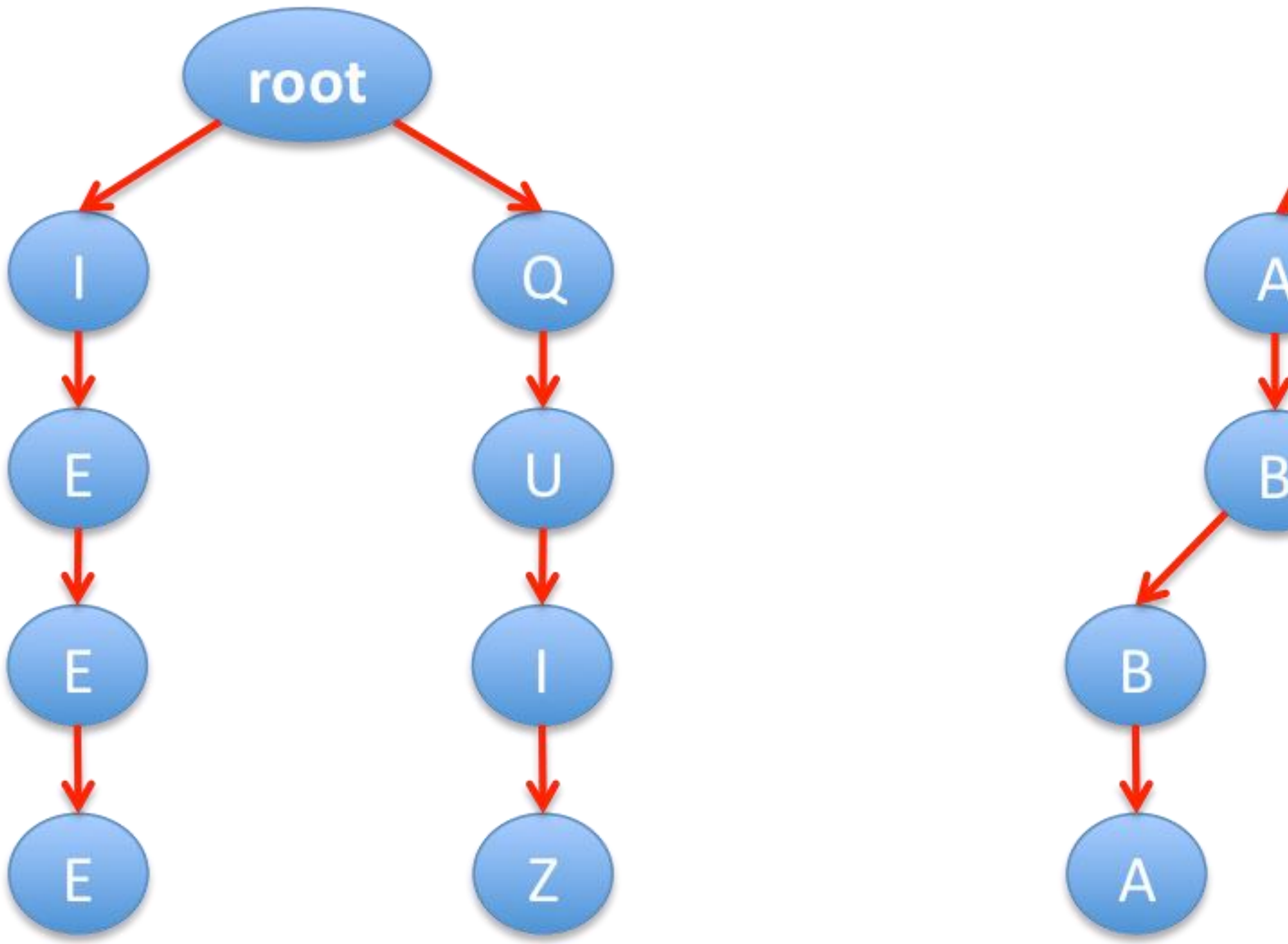
Playing with Tries

Mike has always been fond of playing with words. Recently, he learned programming and started to write programs to find all sorts of things related with texts. However, his programs started to run very slowly when the number of words was very high. Luckily, he found a beautiful data structure to help him called Trie.

Mike uses a Trie like a tree where each edge is labeled with a letter. To build a Trie, Mike first creates a root node. Then, for each word he wants to store in the Trie, he descends from the root node creating nodes whenever necessary. For example, if Mike wants to add “IEEE” to an empty Trie, he follows the steps in the following picture:



Each blue oval represents a node. Each node contains a link to each letter of the alphabet. These nodes can be reused for words with the same prefix. Take "ABBA" and "ABLE" for example. They have the same prefix, so they share the same first two nodes in the Trie as shown the following picture.



On the left, a Trie with the words "IEEE" and "QUIZ". On the right, a Trie with "ABBA" and "ABLE"

The Tries shown above use the 26 letters of the English alphabet. So, each node contains 26 links despite using just one or two of them (e.g. the root node on the left Trie and first node labeled with B on the right Trie use 2 links each, whereas all the remaining nodes on both Tries use only a single link out of the 26 available letters). Each link needs 4 bytes. So, the Trie on the left consumes 9 nodes times 26 links times 4 bytes = 936 bytes. The Trie on the right uses 7 nodes, so it consumes 728 bytes.

Mike is going to play in many words, but hes worried that his Trie will consume too much memory.

Task

Given the size of the alphabet **A** , the maximum length of the words **L** and the number **N** of words to consider, your task is to find what is the maximum amount of bytes necessary to store **N** words in a Trie.

Input

The first line of input contains an integer **T** (1 <= T <= 10^5) the number of test cases.
Then, T lines follow, each containing three integers **A** (1 <= A <= 26), **L** (1 <= L <= 10^5) and **N** (1 <= N <= 10^18), separated by a single space.

Output

For each test case, your program should print one line with a single integer: the worst-case amount of memory necessary.
Note1: The result is guaranteed to fit in a signed 64-bit integer.
Note2: There is a newline character at the end of the last line of the output.

Sample Input

```
4
26 4 2
3 3 10
3 4 3
4 5 101
```

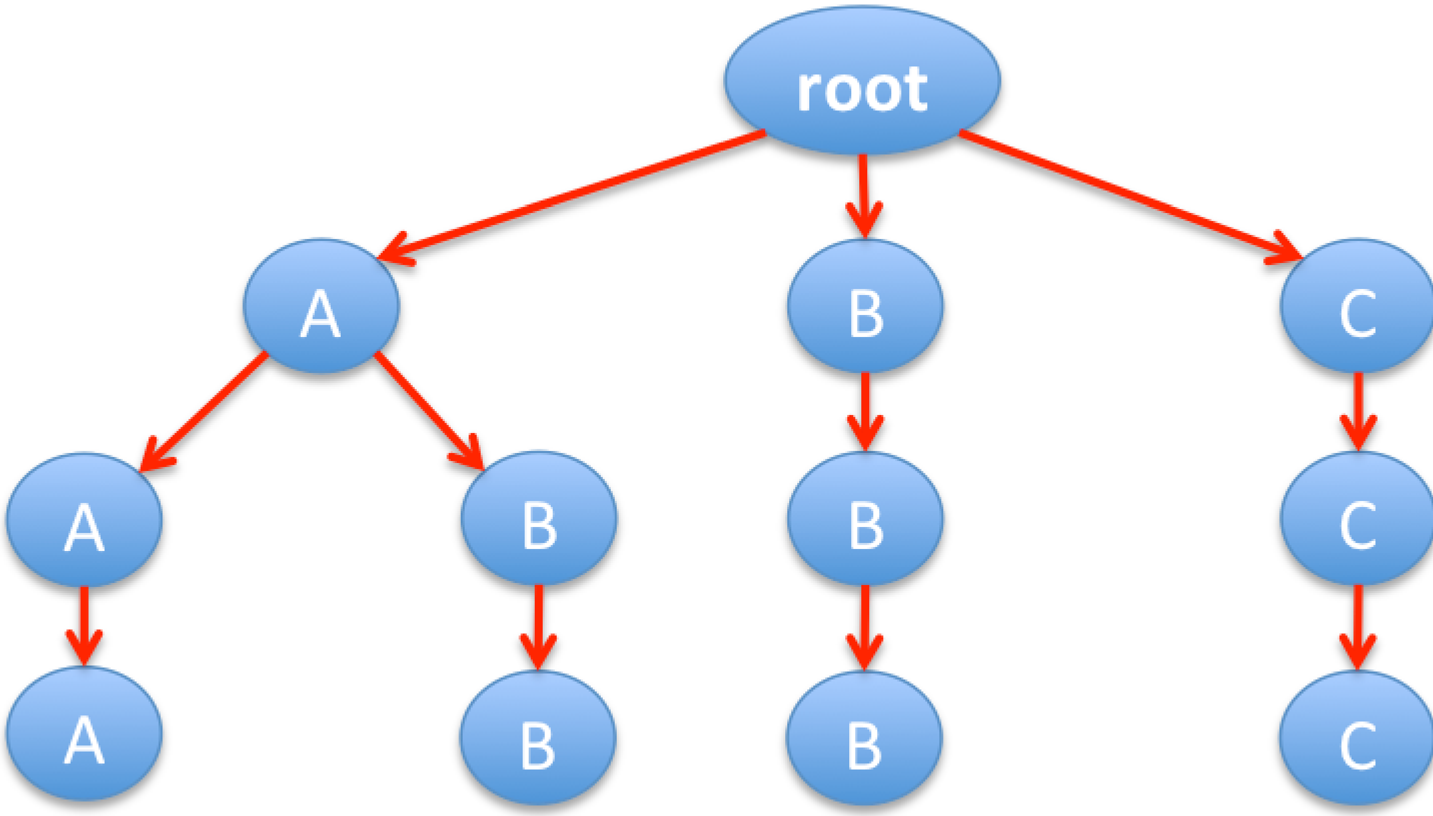
Sample Output

```
936
276
156
4592
```

The first sample input corresponds to the example given in the problem statement.
In the second sample, we have only 3 letters in the Alphabet (e.g. [A, B, C]), 10 words (**N** =10) and a maximum size of 3 letters per word (**L** =3). A possible set of words fulfilling the aforementioned criteria could be the following:

Word 1: A
Word 2: AA
Word 3: AAA
Word 4: B
Word 5: BB
Word 6: BBB
Word 7: C
Word 8: CC
Word 9: CCC
Word 10: ABB

For this example, a Trie that would be suitable for storing these words would look as follows:

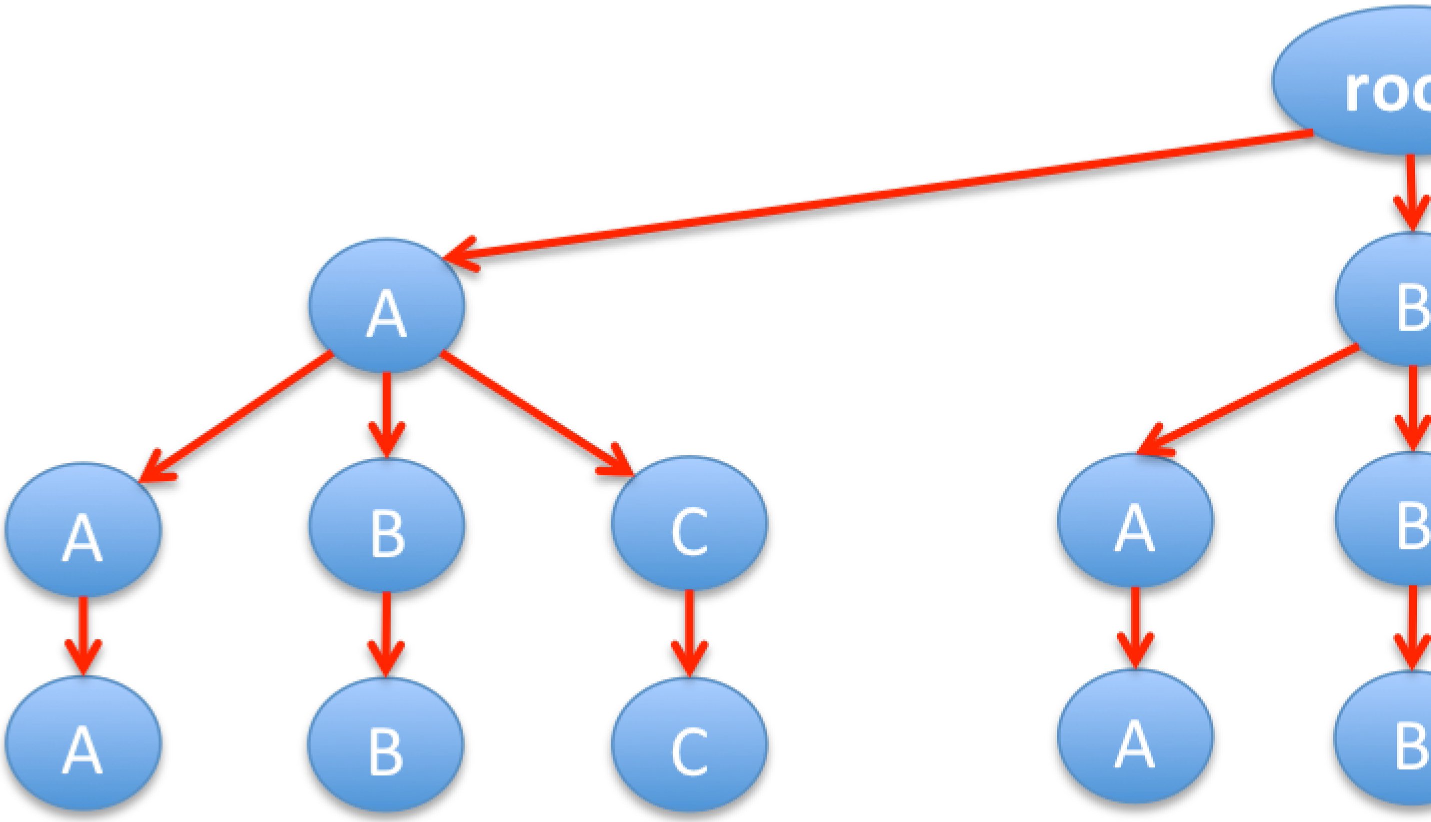


This Trie has 12 nodes so it consumes 12 * 3 * 4 = 144 bytes. However, there might be another set of words which again fulfill the defined criteria for which a bigger Trie would be required to store those words. For example, if the set of 10 words were as follows:

Word 1: AAA
Word 2: ABB
Word 3: ACC
Word 4: BAA
Word 5: BBB
Word 6: BCC

Word 7: CAA
Word 8: CAB
Word 9: CBB
Word 10: CCC

Then, the following Trie would be required, consuming 23 * 3 * 4 = 276 bytes.



Although other Tries would be required for different sets of words, they would all consume less than or equal to 276 bytes. Therefore, the Trie displayed above is one of the worst-case scenarios (i.e. this Trie and possibly several would require at maximum 276 bytes), and for this reason the reported result is 276 bytes.

Play with GCD

Task

Minka is very smart kid who recently started learning computer programming.
He learned how to calculate the Greatest Common Divisor (GCD) of given numbers. The GCD http://en.wikipedia.org/wiki/Greatest_common_divisor of k numbers say [n1,n2,n3… nk] is the largest positive integer that divides all these numbers without any remainder. You may find out more about the GCD and the way it is calculated on the Wikipedia website.
Minka has N (1 <= N <= 10^5) balls and there is a number V (1 <= V <= 10^4) written on every ball. Now Minka has to perform Q queries, and in each query he wants to know the number of possible ways he can choose balls out of the N balls, so that GCD of the numbers written on the chosen balls equals to the number X of each query. Although he already knows the answer for each query, he would still like you to check if you can also find answer to his queries.
Since number of ways can be very large, your program should output the number of ways modulus 10^9+7.
Notes:
1) There can be at most 100 distinct numbers written on N balls.
2) By definition, the GCD is only defined for 2 or more numbers. For this problem, however, we will consider that the GCD of a single number may also defined and in such case the GCD of a single number will be equal to the number itself (i.e. the GCD of 2 is 2. Please refer to the explanation of Sample Input 1 for more details).

Input

The first line of each test file contains an integer N (1 <= N <= 10^5) denoting the number of balls.
The next line contains N space separated integer numbers, each one representing the number written on each of the N balls. The ith number (Vi) corresponds to the number written on the ith ball (1 <= Vi <= 10^4).
The third line contains an integer Q (1 <= Q <= 10^4) representing the number of GCD queries that will have to be performed.
Finally, Q lines follow, each one containing an integer X (1 <= X <= 10^4) corresponding to the GCD of each query.

Output

Your program should output the number of ways modulus 10^9+7 that balls can be drawn from the set, so that their GCD equals the number X corresponding to each query.
Note: There is a newline character at the end of the last line of the output.

Sample Input 1

```
5
2 3 5 6 6
2
2
5
```

Sample Output 1

```
4
1
```

Explanation of Sample Input 1

We have 5 balls in the set, labeled with numbers [2, 3, 5, 6, 6] respectively. For the first query (X=2), there are in total 4 (distinct) ways by which we may choose balls so that their GCD equals 2, meaning:
a) {1, 4} (i.e. ball 1 labeled with number 2 and ball 4 labeled with number 6)
b) {1, 5} (i.e. ball 1 labeled with number 2 and ball 5 labeled with number 6)
c) {1, 4, 5} (i.e. ball 1 labeled with number 2, ball 4 labeled with number 6 and ball 5 labeled with number 6)
d) {1} (i.e. ball 1 labeled with number 2 since according to our definition of GCD, the GCD of 2 would equal 2)

Regarding the second query (X=5), there is only one way to choose balls so that their GCD equals 5, which is to choose only ball 3 (labeled with number 5).

Sum it up!!!

Task

Minka is very smart kid who recently started learning computer programming.
His coach gave him a cyclic array A having N numbers, and he has to perform Q operations on this array.
In each operation the coach would provide him with a number X. After each operation, every element of the cyclic array would be replaced by the sum of itself and the element lying X positions behind it in the cyclic array. All these replacements take place simultaneously.
For example, if the cyclic array was [a, b, c, d], then after the operation with X = 1, the new array would be [a+d, b+a, c+b, d+c].
He needs to output the sum of the elements of the final array modulus 10^9+7.
He made a program for it but it's not very efficient. You know he is a beginner, so he wants you to make an efficient program for this task because he doesn't want to disappoint his coach.

Input

The first line of each test file contains a integer N (1 <= N <= 100000).
The next line contains N space separated integers which represent the elements of the cyclic array (1 <= Ai <= 10^9).
The third line contains a integer Q (0 <= Q <= 1000000) representing the number of operations that will be applied to the array.
Finally, Q lines follow, each one containing an integer X (0 <= X < N).

Output

Your program should output to the standard output stream the **sum of the elements of the final array modulus 10^9+7.**
Note: There is a newline character at the end of the last line of the output.

Sample Input 1

```
5
1 2 3 4 5
2
1
0
```

Sample Output 1

```
60
```

Explanation of Sample Input 1

- After the 1st operation (X = 1), the array would be [1+5, 2+1, 3+2, 4+3, 5+4] = [6, 3, 5, 7, 9]
- After 2nd operation (X = 0), the array would be [6+6, 3+3, 5+5, 7+7, 9+9] = [12, 6, 10, 14, 18]
- Thus the correct answer would equal to = (12+6+10+14+18) % (10^9+7) = 60

Sample Input 2

```
5
1 2 3 4 5
0
```

Sample Output 2

```
15
```

Palindrome

Let us consider the latin alphabet made of 26 characters (letters from 'a' to 'z').
We call word an ordered sequence of characters of arbitrary length.
Given a word w, we call extracted word v from w a word obtained by deleting some characters in w.
For example, "abcd" is an extracted word from "sbanrpsobtspcerudoo".
A palindrome is a word that is symmetric: it can be read indifferently from left to right or right to left. For example, "abccdccba" is a palindrome.

Task

Our goal is to find the length of one of the longest extracted palindrome from a given word w. Of course if w is a palindrome then the answer should be the length of w, and the answer is always greater or equal than 1.

Input

The input given your program is a one line string containing only lowercase latin alphabet letters (no space), followed by a newline character. The length of this input string will never exceed 2000 characters.

Output

The output your program should write is the length of one of the longest extracted palindrome from the input, **followed by a newline character** .

Sample Input 1

```
lukeiamyourfather
```

Sample Output 1

5

Sample Input 2

anynontrivialpropertyofrecursivelyenumerablelanguagesisundecidable

Sample Output 2

21

Grand Integer Lottery

The Integer lottery company is conducting a very special lottery event for the lottery enthusiasts. The scheme for this special lottery is little different from normal lotteries. In particular, there is a sequence of integers called " *lottery sequence* " ranging between S and E which will be generated according to the following rule:

- At first the lottery company decides on a number N representing the total amount of positive integers that a player chooses [i.e. n1, n2, …. nN]. Then the player choose these N positive integers.
- Based on the above user-picked integers, the lottery company generates the *lottery sequence* as follows: For any given integer M in the range [S, E] (inclusive of S and E), M will be in the lottery sequence if at least one user-picked integer when considered as string occurs as a contiguous block in M. In the *lottery sequence* those picked integer Ms are in the sorted order.

For an example: S=1, E=35, N=2, n1=3 and n2=11, then the generated *lottery sequence* would be as follows:
[3, 11, 13, 23, 30, 31, 32, 33, 34, 35] Comprised of all the integers in the range [1, 35] that contain the strings 3 or 11 or both.

After the *lottery sequence* has been generated, the lottery picks the *winning number* using the given *winning index* of the sequence. First integer of the sequence has index 1. For the example above, if the lottery company picked the 5th index as the *winning index* , then *winning number* would be 30 (i.e. the 5th integer of the lottery sequence).

Task

The task in this problem is to find and print the *winning number* of the lottery for the given set of inputs.

Input

The format of the input is as follows:
S E P N
n1
n2

...
nN

The first line of the input consists of 4 space separated positive integers which represent:
S The minimum value from which the *lottery sequence* will be generated
E The maximum value from which the *lottery sequence* will be generated (1 <= S <= E <= 10^6)
P The *winning index* (1 <= P <= 10^6)
N The amount of positive integers that a player picks (1 <= N <= 18)

Then, N lines follow, each one ending with a newline character, representing the N positive integers that were selected by the player. Each of the user-picked integers will consist up to 18 digits (i.e. 1 <= The number of digits in any user-picked integer <=18). Also, for each number it holds true that it begins with a nonzero digit.

Output

Your program should print the *winning number* to the standard output. If no such number exists, then the output should be:
DOES NOT EXIST
Otherwise, the program should print the *winning number* e.g.:
163
Note: There is a newline character at the end of the last line of the output.

Sample Input 1

```
1 10000 4 2
62
63
```

Sample Output 1

```
163
```

Explanation of Sample 1

In this example the user selects 2 positive integers n1=62 and n2=63. Based on this selection, the *lottery sequence* would look like [62, 63, 162, 163, 262, 263, ...]. Since the lottery has picked the *winning index* P=4, the program should output 163 as the *winning number* .

Sample Input 2

```
1 10000 999999 2
62
63
```

Sample Output 2

```
DOES NOT EXIST
```

Rob to the Big Bank

Today is your great day; you and your team have everything ready for one of the biggest hits of your life by launching a heist to the ?Big Bank?. You and your partners manage to take control over all the hostages and guards without many casualties, everything goes smoothly and you are ready to start drilling the big vault for the biggest pay of your life.
Your friend Dallas starts the drill, which unfortunately hangs multiple times during the process (that what happens when you buy cheap tools for a big robbery!) Making the robbery take more time than expected but still the drill manages to open the vaults thick lock.
Unfortunately, something goes wrong and after opening the door, an alarm safety triggers which starts to send an immediate response to the authorities about the breach of the vault. Everyone now knows that the robbery is taking place and you realize that there is not so much time left before the police shows up at your doorstep with a bulldozer and a swat team.
Time is crucial and luckily, you come prepared for this problem and you put in practice your tech skills and run a program in order to help you decide which items you need to carry in order to gain the highest profit based on your total teams load capacity.
Of course, as you would expect a large amount of money is in game as the vault is ?endless?; if not why would you want to rob the ?Big Bank? in the first place? As a good robber and programmer you know that your program needs to be fast enough to give you a solution in a reasonable amount of time so you also give **priority** to the items stored in the vault based on their profit **Ratio = value/load** (you want to keep the items with highest value and have less load).

Task

Your task involves implementing a program that will help the robbers choose how many values they need to grab in order to maximize the revenue of their robbery and the items you need to get from the vault in order to achieve this. You will also show how much share each robber gains from this hit based on what they can carry after the revenue has been decided.

Input

The information for the program will be passed through standard input. The first line will contain two positive single space separated integer values **N** and **M**, corresponding to the number of robbers involved in the heist and to the load capacity of each robber (all robbers will have the same load capacity equal to **M**). .
Then it will follow multiple lines with details of each valuable stored in the vault with their respective load and value, in the following format:

[item name],[whole load amount],[value for whole amount]
Example: *Diamond,2000000,200 : the 200 is the value for the whole amount of 2000000 Diamonds*
The list of elements of the bank will finish with the keyword END in the last line of input.
Note: There will be at maximum 5 items per input test case.

Output

The format of the solution will be in the form of multiple lines where the first lines have the following structure:
[item name],[number of times taken],[total aggregated load of this item],[total aggregated value of this item].
The items in the output string should be listed in alphabetical order (from A to Z) based on the name of each item (for example, if the items chosen to be taken are Jewelry, Money, and Diamond, then tehy should be ordered as Diamond, Jewelry and then Money).
Then, the next line should report the total load of all items in solution and the total value of these items, in the following format:
[total load of all items in solution],[total value of all the items in solution]
Finally the last line of the output should report the split share that each robber gets from the heist depending on the total number of robbers involved. The format of this line should be as follows:
Each robber gets: [value]
(Please note the space after the : and before the [value]).
The [value] for this last line should be **rounded to 2 decimal digits.**
Note: There is a newline character at the end of the last line of the output.

Sample Input 1

```
4 8089000
Diamond,2000000,200
Ruby,3500000,400
Gold,10000000,1000
Jewelry,1500000,160
Money,1000000,100
END
```

Sample Output 1

```
Jewelry,2,3000000,320
Money,1,1000000,100
```

```
Ruby,8,28000000,3200
32000000,3620
Each robber gets: 905.00
```

Sample Input 2

```
5 6471200
Jewelry,995756,962
Money,392852,13
Ruby,564827,1388
Gold,844392,854
Diamonds,800830,584
END
```

Sample Output 2

```
Ruby,57,32195139,79116
32195139,79116
Each robber gets: 15823.20
```

Time to play some DOTA 2!

You and a group of friends decide one afternoon to hang around together in your friends garage for an intensive session of LAN party. One of your friends suggest to play a game of DOTA 2, a popular game present in the most prestigious e-sport events, in particular your friend suggests this game after been able to watch this years “The International” DOTA 2 tournament which gathers the best teams in the whole world.
“DOTA 2 is a multiplayer online battle arena game, with the gameplay focused around combat in map divided in two fronts where teams of 5 players play against each other. The player may command a single controllable character called a "Hero", which is chosen from a selection pool of heroes. Each Hero begins the match at level one, but may become more powerful by leveling up by accumulating experience through combat. The Hero's methods of combat are influenced by its primary property, which can be Strength, Agility, or Intelligence.”
The night is long and you and your friends played for hours and after that long gaming session you and your friends come to the conclusion that this game is so fun and challenging that you start thinking of creating a competitive team with your friends. You are so excited with the idea that you start to plan your first steps into competitive play and you realize that you need some sort of feedback of your actual gameplay in order to improve and become one of the best. Unfortunately, you realize this problem involves some technical skills and luckily, you are fortunate to know that one of your friends is a good proficient engineer in programming which also likes DOTA 2 and decides to give you a hand to devise your own Hero Coacher.
He states that he has the perfect way to identify which heroes a player needs to select depending on its previous history from a small selection. He defines the quality of a hero for that player, as qi = fi/zi.
fi = winner ratio percentage which is proportional to win:loss ratio of all the games played by that player **rounded down to the nearest whole number**
zi = world wide rank, which in our case it is assumed as follows:
*“The list of proposed heroes will always be given in the order of their popularity, this means that the first hero is picked more than the second hero, the second more than the third, etc.. Briefly speaking, we can see that further down this hero is in the list, the less popular he is.
In order to simplify things; the value zi can be mapped as the relative popularity based on its position in the list (i'th position) which will be proportional to 1/i.”*
Example: *“If the hero Spectre with 34:22 ratio is located in the position 3 of the list, its world wide rank is proportional to 1/3 and its winner ratio will be 60 (i.e. 34/(22+34) = 0.607 = 60.7%, therefore his winner ratio will be 60, i.e. 60.7=60 when rounded* ***down*** *to the nearest whole number) so q3 = 60/(1/3) = 180”*
In addition, you are capable of identifying a players affinity for a certain type of hero from the Strength, Agility and Intelligence areas on the pool of the selected group of heroes that you give as a solution.

Task

Your task is to select from a pool of heroes and game plays statistics of the player, a subset of desired heroes that the player excels most based on the quality of a hero. In addition, you should give out the affinity of the hero type of that player based on the small pool asked previously for.

Input

The first line of input contains two integers n and m (1 <= n <= 50000, 1 <= m <= n), the number of heroes available, and the number of heroes to select. Then follow n lines. The ith of these lines contains a string with the hero name, the hero type and the win:loss ratio with that hero, separated by commas.

Output

Output a list of the m heroes with the highest quality qi, in decreasing order of quality. If two heroes have the same quality, give precedence to the one appearing first on the list.
After that, you should print the affinity in the order *Intelligence, Strength, Agility* for each type of hero as a percentage **formatted down to 2 digits** on the selected group of heroes.
Note: There is a newline character at the end of the last line of the output.

Sample Input 1

```
4 2
Silencer,Intelligence,83:80
Tiny,Strength,94:76
Spectre,Agility,34:22
Dazzle,Intelligence,60:89
```

Sample Output 1

```
Spectre
Dazzle

This set of heroes:
Contains 50.00 percentage of Intelligence
Contains 0.00 percentage of Strength
Contains 50.00 percentage of Agility
```

Sample Input 2

```
15 3
Lina,Agility,91:29
Necrophos,Intelligence,93:36
Lycan,Strength,95:54
Skeleton King,Strength,20:33
Chaos Knight,Strength,67:75
Leshrac,Intelligence,16:63
Magnus,Strength,31:19
Anti Mage,Agility,55:99
Rubick,Intelligence,13:33
Clinkz,Agility,9:95
Drow Ranger,Agility,12:12
Bane,Intelligence,89:16
Spirit Breaker,Strength,63:20
Templar Assassin,Agility,74:57
Spectre,Agility,72:49
```

Sample Output 2

```
Bane
Spirit Breaker
Spectre

This set of heroes:
Contains 33.33 percentage of Intelligence
Contains 33.33 percentage of Strength
Contains 33.33 percentage of Agility
```

Road Trip

Vangelis the bear is planning a long road trip during which he wants to visit all the landmarks along a path of length L. The tank of his car can take up to F units of fuel and for every unit of distance covered his car consumes a unit of fuel.
Vangelis knows how far each of the N gas stations are from the beginning of the path and the price per fuel unit each station offers.
At the starting point he has T units of fuel in his car.

Task

Write a program that will accept the above information and will calculate the minimum amount of money Vangelis needs to spend. If the journey is impossible to make, it should print -1.

Input

The first line will contain an integer M (1 <= M <= 10) denoting number of Test cases.
The next line will contain four space separated integers:
N (0 < N < 50001): The total number of gas stations
F (0 < F < 1000001): The units of fuel Vangelis's car can take
T (0 <= T <= F): The units of fuel Vangelis's car has at the beginning of the trip
L (0 < L < 1000000001): The path length of the landmarks he plans to visit
Each of the following **N** lines will contain two integers: the first one, **D_i** (0 <= D_i <= L) corresponds to the distance of the station from the starting point, and the second one, **C_i** (1 <= C_i <= 1,000,000) represents the cost per fuel unit for that station.

Note: You may assume that the trip will be on a straight line where all gas stations are spread on this line at the positions specified by their D_i values.

Output

There should be M lines with the minimum amount of money to be spent per test case or with -1 in case the trip is not feasible.
Note: There is a newline character at the end of the last line of the output.

Sample Input 1

```
1
4 20 6 34
4 40
18 15
10 7
20 12
```

Sample Output 1

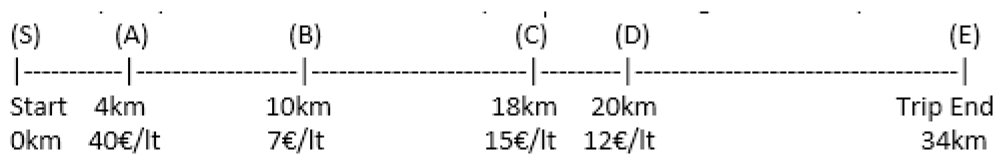
```
348
```

Explanation of Sample Input 1

This sample contains only 1 test case (M=1.) The second line of the input is 4 20 6 34 which means that:
a. There are in total N=4 gas stations on the route
b. The (max) fuel capacity of Vangelis car is F=20 liters
c. The tank currently has T=6 liters of gas
d. Vangelis wants to travel L=34 kms in total

Then the details for the 4 gas stations are provided in the form Di Ci, where Di is the distance of this gas station from the starting point and Ci is the cost per liter of gas:
4 40
18 15
10 7
20 12

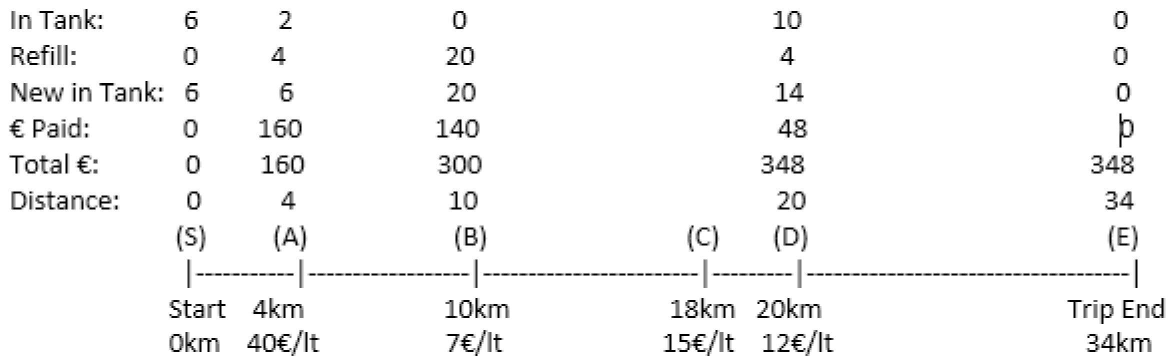
For simplicity assume that the whole trip is done in a straight line as depicted below:



Obviously Vangelis does not have enough fuel for all 34 kms, so he needs to refuel. The cheapest gas station is the one labeled (B) above, however Vangelis does not (initially) have enough fuel in his tank to reach (B), since B-S = 10 and he has T=6. So he needs to add an extra

4 liters from gas station A, so that he can the make it until gas station B to get as much (cheap) as he can in order to make his 34 km journey. Thus he pays (i) 4lt * 40€/lt = 160€ and now he can make it until (B). Since until this moment he has only traveled 10 kms, he needs gas for another 34-10=24kms. Normally he would want to refuel his car with 24 liters (since B is the cheapest gas station) but since his (max) fuel capacity is F=20 liters he will only take 20 liters and thus pay (ii) 20 lt * 7 €/lt = 140€. He knows however that up to point (B) he has only traveled 10kms and he needs to travel another 24kms to reach his goal, whereas he has gas for 20kms. So he would have to stop at a later gas station (after he has traveled at least 4kms) to refuel another 4 liters of gas so that he could complete the whole 34 kms journey. Since he now has quite some gas, he may decide whether he wants to refuel at (C) or at (D) and since (D) is cheaper, it is more than 4kms away from (B) and is within reach (based on his gas in the tank) he will choose to refuel another 4 liters at (D) and thus pay (iii) 4lt*12€/lt=48€). After that he can successfully reach the end point of his trip.

See also the revised illustration below: (1st line is the current fuel capacity, 2nd line represents how much gas he refueled, 3rd line is fuel capacity after refilling, 4th line the money spent for the refilling, 5th line the total amount of money spent so far and 5th line the total distance covered so far):



```
In Tank:        6      2          0                       10               0
Refill:         0      4          20                      4                0
New in Tank:    6      6          20                      14               0
€ Paid:         0      160        140                     48               0
Total €:        0      160        300                     348              348
Distance:       0      4          10                      20               34
               (S)    (A)        (B)            (C)      (D)              (E)
                |-----------|-----------------|-----------------------|---------|-----------------------------------|
              Start   4km        10km          18km     20km           Trip End
               0km   40€/lt      7€/lt         15€/lt   12€/lt           34km
```

In total he has spent: (i) 160€ + (ii) 140€ + (iii) 48€ = 348€ as reported by the test case output.

Run me

This problem seems quite easy: it seems that we are giving you the answer… All you have to do is supply the output of **our** program on the given input.

Input

A string of characters, ending with a dot ('.').

Output

Your program should print the output of an MS-DOS 8086 assembly program p.com on the given input.
Our program (in hex dump) is:

```
BF 00 04 BE C0 00 56 31 C9 B4 00 CD 16 3C 2E AA
E0 F7 F7 D1 29 D2 89 CD 5B 53 FE 07 75 03 43 EB
F9 BF 00 02 89 F9 89 F8 F3 AA 89 FE AC 89 C3 FE
07 80 FB 2E 75 F6 FE 0F 5E 56 89 E9 AC 89 C3 FE
0F 7C D5 E2 F7 42 5E 56 89 E9 F3 A6 75 CA 5D 92
D4 0A E8 00 00 86 C4 04 30 CD 29 C3
```

Note: There is a newline character at the end of the last line of the output.

Sample Input 1

```
5.
```

Sample Output 1

```
01
```

Sample Input 2

34.

Sample Output 2

```
02
```

Right-Angled Triangle

Vangelis the bear wants to place a set of right-angled triangles on a Cartesian coordinate system in such way that the legs (catheti) a and b of the triangles are not parallel to the X and Y axes and all the triangle vertices are coordinate pairs of integers.

Task

Write a program that will accept the lengths of the triangle legs and will print "TRUE" if the triangle can be placed following the above conditions and print "FALSE" if it fails the above conditions.

Input

The first line will contain an integer **N** ($0 < N < 11$), the number of triangles to study.
Each of the following **N** lines will contain two space separated integers **a,b** ($0 < a,b < 1001$), corresponding to the lengths of the triangle legs.

Output

There should be **N** lines, on each line there should be the word "**TRUE**" or the word "**FALSE**" (without the quotes).
Note: There is a newline character at the end of the last line of the output.

Sample Input 1

```
3
1 1
5 5
5 10
```

Sample Output 1

```
FALSE
TRUE
TRUE
```

Explanation of Sample Input 1

Triangle 2: Can be placed on the following coordinates:
(2,1),(5,5),(-2,4).
Triangle 3: Can be placed on the following coordinates:
(-10,4),(-2,-2),(1,2).

Coffee Empire

Patras is a city in Greece especially known for one thing, coffee shops. Wherever you go, you will find a coffee shop to buy a coffee "on the go", or to sit and drink a hot or cold one. But let's meet our investor, Kathrin.
Kathrin wants to invest her money and build a new coffee shop. However, to cope with the high competitiveness, she is trying to find the optimal place to build it. Towards this direction, she collected some really detailed statistics. These statistics include the number of coffees that could be bought per corner (for all corners of town) on any given day of week. This number is calculated based on the number of people passing by each corner. The statistics also suggest that all shops sell coffees for the same price, meaning **one euro per coffee** , and that each shop, in order for it to be able to cover the expenses for warming up the appliances and for the employee that serves the customers, it should gain at least 20 euros for any day that is open (otherwise it should remain closed).
Based on these, Kathrin found a simple way to calculate the income for any given corner, which she plans to use in order to find the best spot to open her coffee shop. Let's have the following example with named corners:
A B C
D E F
G H I
First we define the notion of **adjacent corners** : the adjacent corners of a specific corner are the ones that are positioned in cross—like shape with regards to the corner at question. For example, the adjacent corners of corner E would be the ones labeled with B, D, F and H whereas the adjacent corners for corner C would be the ones labeled with B and F.
Based on the definition above and according to Kathrins plan, the expected future income ***for our coffee shop*** and for a given day and a given corner should be calculated according to the following rules:

- The daily income for that corner would equal to the respective value according to the detailed statistics **plus** any additional income that might derive from its adjacent corners (see rule below) .
- If we were to build our coffee shop at a particular corner, then we should also take into account additional expected daily income deriving from coffees that may be sold due to traffic from the ***adjacent corners*** of our corner. In particular, for every adjacent corner of the corner at question that **a)** has a daily income (**as reported by the statistics**) greater or equal to 5 Euros **and b)** has **exactly 3 coffee shops** already located in it, we can safely assume **an extra daily income of 1 Euro** due to that adjacent corner. If either or none of the conditions are not met for an adjacent corner, then we should not expect additional income from it.
- The income corresponding our shop for a given day and corner, will equal the daily income for that corner (**as reported by the statistics plus any additional income based on its adjacent corners**) divided by the overall number of coffee shops located at this corner (note that this value is not always guaranteed to be an

integer). For example, if there already exist 2 coffee shops at corner E and the daily income for this corner according to the statistics and the adjacent corners of E equals 27 euros, then we should expect that the daily income for our shop will be 9 euros, as the overall amount of 27 euros will be equally split to all three coffee shops of this corner (also considering our own shop in case we decide to build it at that specific corner).

- In order for our coffee shop to be open during a given day of the week, the total income corresponding to our coffee shop for that day should be **greater or equal** to its daily expenses (i.e. **20 Euros**). Otherwise, there is no point opening our coffee shop during that day and thus both income and expenses for that day would equal zero.
- According to the town hall rules, it is **forbidden to install more than 3 coffee shops per corner** .

Task

Your task is to develop a program that can find the position that is best for us (i.e. the position that makes us the most profit **on a weekly basis**) to set up our new coffee shop. Can you help us?

Input

Your program will receive the following input from the Standard Input Stream:
In the first line, the width (W<=100) and height (H<=100) of our town, which we can assume that is a perfect grid.
Then H lines will follow each one describing the number of coffee shops already located at each corner. Each line will be composed of a character belonging to the [-, L, M, H] alphabet followed by an asterisk (*), with the exception of the character corresponding to the last corner of each line, which will be followed by a newline character. A dash "-" means "no coffee shops", "L" means only one coffee shop, "M" means two and "H" means three. For example, the first line of the provided sample case indicates that there are no coffee shops located at the first corner (X:1, Y:1), three coffee shops on the second corner (X:2, Y:1) and no coffee shops on the third corner (X:3, Y:1).
Then seven [W, H] blocks of lines will follow, each one corresponding to the statistics of each corner for each day of the week. In the first line of each block, the name of the day will be given followed by the number of coffees that will be sold per corner. Again, the number of coffees sold per corner will be separated by an asterisk (*) with the exception of the number corresponding to the last corner of each line, which will be followed by a newline character.

Output

Your program should output to the Standard Output Stream the coordinates where we should place our new coffee shop. The coordinates are considered to start from 1 and should be reported in the [X, Y] order, where X corresponds to a column and Y to a line of the town grid. These two values should be separated by a single space character. If there are more than one corners with as equally as optimal income, then the first should be printed. In case, there is no such a place, then the output “-1 -1” (without the quotes) should be printed on the screen.
Note: There is NOT a newline character at the end of the output stream

Sample Input 1

```
3 3
-*H*-
M*M*-
H*-*L
Monday
30*1*1
20*13*1
0*0*0
Tuesday
0*2*2
0*1*1
1*1*1
Wednesday
0*0*2
1*1*3
0*0*1
Thursday
0*1*1
0*1*0
0*1*1
Friday
0*0*0
0*2*3
0*1*1
Saturday
0*2*0
1*3*0
0*1*2
Sunday
0*2*3
0*1*0
1*0*0
```

Sample Output 1

```
1 1
```

Rajus Training

Raju was accepted at the Computer Science Department of a leading University and he is now interested in becoming a winner at IEEEXtreme 8.0. However his programming skills are not ?sharp? enough and thus he asked the help of Rano, a senior year student of the same department.
Raju has analyzed all subjects of study that has to mastered before he becomes proficient in programming and has discovered that there are some constraints in the way that these subjects should be studied (i.e. certain subjects may be studied before some others). Rano, who is more experienced, volunteered to organize these subjects according to their constraints and provide Raju with a plan of study that will be consistent to the provided constraints. Can you verify that Ranos plan of study indeed satisfies all constraints?

Task

Given a number of study subjects (N) and some constraints (M), your task is to evaluate a given study plan and assert whether it satisfies all constraints or not.

Input

In the first line of the input, two space-separated positive integer values [N, M] will be provided (1 <= N <= 1000; 0 <= M <= 100000). The first number (N) corresponds to the number of study subjects whereas the second one represents the number of constraints characterizing these subjects.

Then, M lines will follow, each one containing two space-separated positive integer values [S, U] (1 <= S <= N; 1 <= U <= N). Each of these two integers corresponds to a study subject and when both numbers are considered as a pair,they describe a constraint (for example the pair [3 4] would mean that subject 3 must be studied **before** subject 4).

The final line of the input contains a sequence of space-separated positive integers, each of which corresponds to one of the N study subjects. This line represents the study plan that was proposed by Rano.

Output

Your program should output to the standard output stream a single line containing the word:
YES (if the proposed study plan fulfills **all** the provided constraints) , or the word:
NO (if the proposed study plan fails to fulfill **even a single constraint**)
Note: There is a newline character at the end of the last line of the output.

Sample Input 1

```
5 6
1 3
1 4
3 5
5 2
4 2
1 2
1 3 4 5 2
```

Sample Output 1

```
YES
```

Assembly Simulator

Assembly Language is a low-level programming language that is specific to a particulars computer architecture. Each command has a specific structure:

label COMMAND OPERANDS

The commands are represented by mnemonics and perform low-level arithmetic and logical operations generally computed by an Arithmetic-Logic-Unit (ALU). Operands include constants, registers, and addresses that can trigger operations. Labels are used to mark the line of a command so that the ALU can jump or branch to that location to implement higher-level operations such as an if-statement or loop.

Task

Write an assembly language emulator for the Xtreme-8000 processor. The processor consists of an ALU and a register to store comparisons (only set using the COMP command). There are no accumulators, temporary data registers, or temporary address registers. The processor has a maximum of 256-byte memory that it can address through its commands. The full list of commands is:

`PRINT A1`	Print the hexadecimal byte stored at memory location A1.
`PRINT A1,A2`	Print the hexadecimal bytes stored in locations A1 to A2 (both addresses are inclusive).
`MOVE #N,A1`	Move constant N to memory location A1.
`MOVE #N,(A1)`	Move constant N to the dereferenced address A1.
`MOVE (A1),A2`	Move the dereferenced address A1 into A2.
`MOVE (A1),(A2)`	Move the dereferenced address A1 into the dereferenced address A2.
`MOVE A1,(A2)`	Move the byte at A1 to the dereferenced address A2.
`MOVE A1,A2`	Move byte from memory location A1 to A2.
`ADD #N,A1`	Add constant N to the value stored at A1. Result stored in A1.
`ADD A1,A2`	Add value from A1 to A2. Result stored in A2.
`SUB #N,A1`	Subtract constant N from the value stored at A1 and store the result in A1.
`SUB A1,A2`	Subtract value A1 from A2, and store the result in A2.
`AND #N,A1`	Logical AND between constant N and A1. Result stored in A1.
`AND A1,A2`	Logical AND between values at A1 and A2. Result stored in A2.
`OR #N,A1`	Logical OR between constant N and A1. Result stored in A1.
`OR A1,A2`	Logical OR between values at A1 and A2. Result stored in A2.
`XOR #N,A1`	Logical XOR between constant N and A1. Result stored in A1.
`XOR A1,A2`	Logical XOR between values at A1 and A2. Result stored in A2.
`COMP #N,A1`	Compare constant N to A1. Result stored in the processor to use in the following branch operation.
`COMP A1,A2`	Compare values stored at A1 and A2. Result stored in the processor to use in the following branch operation.
`BEQ label`	Branch to label if result of comparison is equal.
`BNE label`	Branch to label if result of comparison is not equal.
`BGT label`	Branch to label if result of comparison is true (A1 > A2).
`BLT label`	Branch to label if result of comparison is true (A1 < A2).
`BGE label`	Branch to label if result of comparison is true (A1 >= A2).
`BLE label`	Branch to label if result of comparison is true (A1 <= A2).

Input

The input will contain:

- 0 <= Size of memory in bytes <= 0xFF
- Program to execute

- Each line has an optional label, a command, and a list of comma separated operands
  (as needed for each command).

Output

Output the result of any print statement.

Note: There is a newline character at the end of the last line of the output.

Sample Input 1

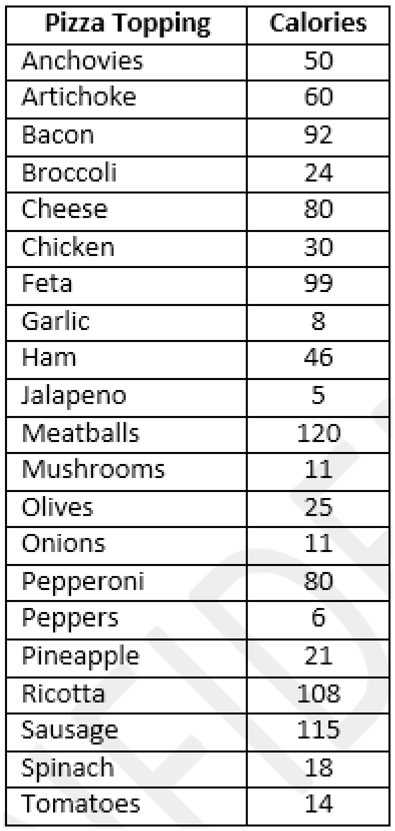
```
0F
PRINT 00,0F
MOVE #FF,00
PRINT 00,0F
MOVE 00,01
PRINT 00,0F
ADD 00,02
PRINT 00,0F
ADD 01,02
PRINT 00,0F
```

Sample Output 1

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
FF FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00
FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00 00
FF FF FE 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Resist-our Pizza

Pizza is one of the top food choices for the IEEE Xtreme Programming Competition in North American universities. Although delicious, there are 270 calories in each pizza slice, plus additional calories for toppings as described in the table below.



Pizza Topping	Calories
Anchovies	50
Artichoke	60
Bacon	92
Broccoli	24
Cheese	80
Chicken	30
Feta	99
Garlic	8
Ham	46
Jalapeno	5
Meatballs	120
Mushrooms	11
Olives	25
Onions	11
Pepperoni	80
Peppers	6
Pineapple	21
Ricotta	108
Sausage	115
Spinach	18
Tomatoes	14

This problem was designed to help teams select food for the IEEE Xtreme that would provide a more balanced diet and therefore improve their performance in the competition.

Task

Write a program that takes as an input a number of different pizza combinations and estimates the total calories eaten for that meal.

The input should be of the form: number of different pizza combinations, number of slices of each pizza, and toppings selected for each of the slices.

Input

The input will contain:

- 1 <= different pizza combinations <= 100
- For each pizza combination:
  - 0 <= Number of slices of pizza eaten < 100
  - A comma separated list of toppings with at least one topping (no spaces before/after commas).

Output

The output should state, "The total calorie intake is " followed by the number of calories.
Note: There is a newline character at the end of the last line of the output string.

Sample Input 1

```
2 1 Pepperoni 2 Pineapple,Ham
```

Sample Output 1

```
The total calorie intake is 1024
```

Elementary Cellular Automaton

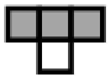
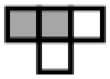
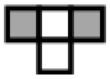
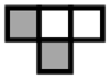
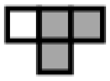
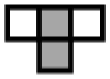
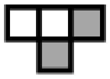
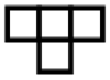
Elementary Cellular Automaton (ECA) is a discrete modeling technique used in science and engineering to study the behavioral patterns that emerge in nature. For more information see http://mathworld.wolfram.com/ElementaryCellularAutomaton.html.

The state of the automaton consists of an array of *N* cells that can take either a TRUE or FALSE state. For each iteration, a new state is computed based on the current state and a set of rules for the automaton.

The new state for cell *i* is calculated based on the previous state of cell to the left (*i-1*), same position (*i*), and to the right (*i+1*). If the state happens to coincide with an edge (e.g., when *i=0*), then the "imaginary" cell is assumed to be FALSE.

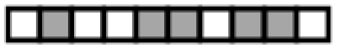
There are 2×2×2=8 possible binary states for the three cells neighboring a given cell as shown in the table below. Furthermore, there are a total of 2^8=256 elementary cellular automata, each of which can be indexed with an 8-bit binary number. This is usually known as a rule. For example, rule 30 (30 base 10 = 00011110 base 2) is shown in the table below. In the diagram, the previous state's left, middle, and right cells are shown in the top row and the second row shows the current cell in the new state being modified.

The rules for this assignment are as follows:

#	Left Cell (i-1)	Middle Cell (i)	Right Cell (i+1)	New Cell (State j+1)	Pictorial Representation
1	TRUE	TRUE	TRUE	FALSE	
2	TRUE	TRUE	FALSE	FALSE	
3	TRUE	FALSE	TRUE	FALSE	
4	TRUE	FALSE	FALSE	TRUE	
5	FALSE	TRUE	TRUE	TRUE	
6	FALSE	TRUE	FALSE	TRUE	
7	FALSE	FALSE	TRUE	TRUE	
8	FALSE	FALSE	FALSE	FALSE	

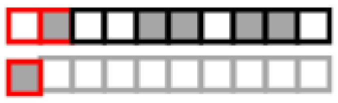
CONFIDENTIAL

For example, consider the case where *N=10* and it is set as follows for state *j* :

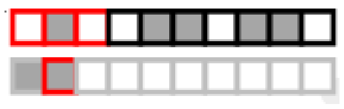


Then, to calculate state *j+1* , we need to iterate through every cell as follows. For each case, the neighbors are highlighted to make it easier to see.

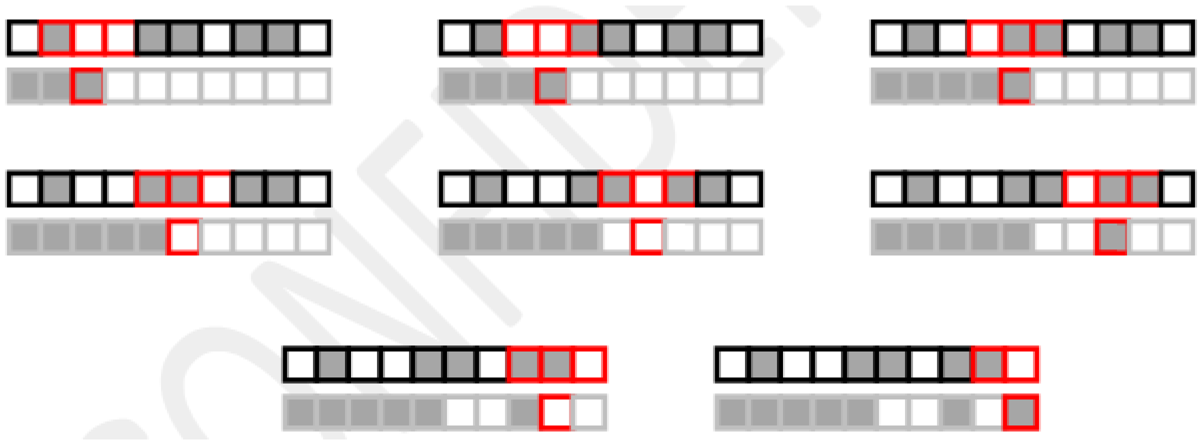
When *i=0* , the previous state is {FALSE, FALSE, TRUE}, which corresponds to rule #7 that says the result should be TRUE. Note that in the previous state, the cell at *i-1* does not exist, so we assume it is FALSE.



Then, we can calculate the next cell at position *i=1* which corresponds to rule #6.



Continuing on, we see the following changes for *i=2* through *i=9* .



Thus, the new state has a different pattern of TRUE/FALSE cells. The process then repeats starting with the new state.

The cellular automaton ends after the cells reach a steady state (no more changes from one iteration to the next) or one reaches a maximum number of iterations.

Task

Write a program that can draw any rule in the elementary cellular automaton given a rule number, integer N for the number of cells, and initial conditions for the cells. The program should output the automaton until either of the two terminating conditions is met.

Input

The input will contain the following single-space separated parameters:

- 1 <= Rule # <= 256
- 0 <= Maximum number of iterations to print < 1000
- 8 <= Number of cells, N <= 64
- 0 <= Integer representing initial condition for N cells <= 2^64 -1
  (where the binary representation shows the TRUE/FALSE state of the N cells)

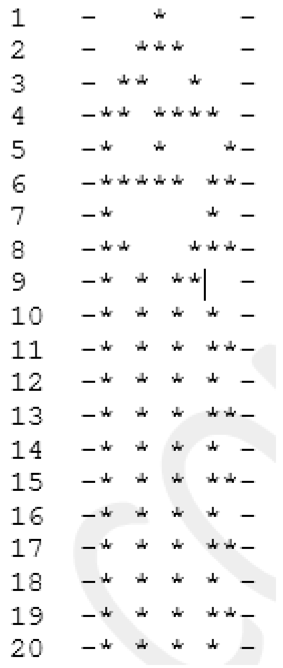
Output

The program should output the iteration number formatted as a left-aligned 3-digit number padded with spaces. Then, each iteration of the automaton should be displayed with cells marked as TRUE shown as an asterisk character (*) and cells marked as FALSE should be a space character ( ). The cells in the state should be surrounded by dashes to make it easier to differentiate when the empty states are.
Note: There is a newline character at the end of the last line of the output.

Sample Input 1

30 20 8 16

Sample Output 1



```
1   -   *    -
2   -  ***   -
3   - **  *  -
4   -** **** -
5   -*  *   *-
6   -***** **-
7   -*     * -
8   -**   ***-
9   -* * **  -
10  -* * * * -
11  -* * * **-
12  -* * * * -
13  -* * * **-
14  -* * * * -
15  -* * * **-
16  -* * * * -
17  -* * * **-
18  -* * * * -
19  -* * * **-
20  -* * * * -
```

Nekops-Nu Sequences

A Nekops-Nu sequence is computed from any sequence of integers by following these rules starting from the first number (left-most number) in the sequence:

- Count the number of *successive* repetitions of the same number.
- Output the count and then the number.
- Repeat steps 1-3 for the next digit in the sequence

For example, given the sequence "1 2 1 1", the Nekops-Nu sequence can be computed as:

- There is 1 instance of the number 1.
- There is 1 instance of the number 2.
- There are 2 instances of the number 1.
- The new Nekops-Nu sequence is "1 1 1 2 2 1"

The same set of rules can be applied to the resulting sequence to generate more patterns. See the sample output for further continuations to the example provided.

Task

Write a program that computes the next K instances of the Nekops-Nu sequence for a given sequence of numbers.

Input

The input will contain:

- 0 <= Number of iterative sequences to compute, K <= 10
- Initial sequence of 1 <= N <= 250
- The numbers in the sequence can have any valid 32-bit integer value.

Output

The output contains one Nekops-Nu sequence per line starting with the initial input sequence and followed by the K sequences computed. The numbers in each sequence are separated by a single

space character.
To make the output look more interesting, add periods to pad all the sequences (before and after) such that they are centered align with respect to the longest sequence computed. Given the example above, the two sequences were “1 2 1 1” (length 7 characters) and “1 1 1 2 2 1” (length 11 characters). Therefore the output should show:

```
..1 2 1 1..
1 1 1 2 2 1
```

If a specific row and the longest row do not line up (because of odd and even numbers of characters), then add an extra dot **before the sequence** as shown in input/output #2, or refer to the following example:

```
.....9999 9999....
......2 9999......
....1 2 1 9999....
1 1 1 2 1 1 1 9999
```

(i.e. if the number of characters is even in a specific row, then there should be an extra dot before the sequence)

Finally, on the last line, output the number of elements found in the last sequence.

Note: There is a newline character at the end of the last line of the output.

Sample Input 1

```
3 1 2 1 1
```

Sample Output 1

```
....1 2 1 1....
..1 1 1 2 2 1..
..3 1 2 2 1 1..
1 3 1 1 2 2 2 1
8
```

Sample Input 2

```
1 1 1 1 1 1 1 1 1 1 1
```

Sample Output 2

```
1 1 1 1 1 1 1 1 1 1
........10 1.......
2
```

The Pipeline

You have to lay a pipeline through the great city X. The city is represented by a grid of N x N cells. Columns are numbered 1 to N from left to right. Rows are numbered 1 to N from top to bottom. In order to lay the pipeline you have to dig in the cells that the pipeline goes through. There is a cost involved with digging up each cell. Your task is to minimize the total cost for digging up cells.

Furthermore pipeline should be laid from left column (column 1) to right column (column N) of the city satisfying the following conditions:

1. Since the city is located in the middle of a dessert it doesn't matter from which row the pipeline starts and ends. i.e. The pipeline should start from any cell of column 1 and should end at any cell of column N.
2. When pipeline enters into the column i (1 <= i < N), the pipeline can be laid to column i+1 using one of following ways: (lets say pipeline enters the column i at row j)
    - Lay the pipeline upward any number of cells (say d) within the city boundary and move to column i+1 at row j-d. i.e. d+1 cells in column i will be digged up
    - Lay the pipeline downward any number of cells (say d) within the city boundary and move to column i+1 at row j+d. i.e. d+1 cells in column 1 will be digged up
    - Lay pipeline to column i+1 through current cell. i.e. Only 1 cell will be digged up in column i

Task

Given digging up costs for each cell in the city, find the minimum cost for laying the pipeline from column 1 to column N.

1 <= N <= 1000
0 <= cost of digging up the cell at column i and row j, c(i,j)<= 1,000,000

Input

First line of the input will contain an integer N, the number of rows and column in city.
Next N lines will represent row 1 to row N respectively.

Each row will have N integers separated by spaces representing costs of column 1 to column N respectively of the corresponding rows.

Output

Output should contain only one integer, minimum cost for digging up the cells that pipeline goes through.
Note: There is a newline character at the end of the last line of the output. If you are using C++, use scanf() instead of std::cin as it can be slower reading input

Sample Input

```
5
1 1 9 1 1
3 1 9 7 1
4 1 9 1 1
5 1 1 1 5
6 1 9 3 1
```

Sample Output

```
9
```

Sample Explanation

Pipeline starts from (1,1).
So pipeline enter the column 1 at (1,1). Moves to column right.
So pipeline enters the column 2 at (2,1). Moves 4 cells downward and move to column right.
So pipeline enters the column 3 at (3,4). Moves to column right.
So pipeline enters the column 4 at (4,4). Move 1 cell upward and move to column right.
So pipeline enter the column 5 at (5,3)
Total cost of the path = 1 + (1 + 1 + 1 + 1) + 1 + (1 + 1) + 1 = 9

Note: Other paths would produce larger answers. For example, if you consider taking the first row across, then the answer would be 1+1+9+1+1=13, which is larger than the shortest path identified.

“K”-value

You are given N integers that are arranged circularly. There are N ways to pick consecutive subsequences of length M (M < N). For any such subsequence we can find the “K”-value of that subsequence. “K”-value for a given subsequence is the Kth smallest number in that subsequence.

Task

Find the smallest “K”-value out of all possible subsequences.
1 < N <= 100,000
1 <= M < N
1 <= K <= M
0 <= any integer in the circle <= 2,147,483,647

Input

First line of the input will contain three integers N, M and K separated by spaces respectively. Second line of the input will contain N integers separated by spaces in clockwise order starting from an arbitrary location.

Output

Output should contain only one integer, smallest “K”-value out of all possible subsequences.
Note: There is a newline character at the end of the last line of the output.

Sample Input

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

Sample Output

```
2
```

Sample Explanation

2nd smallest of subsequence 1, 5, 3 is 3
2nd smallest of subsequence 5, 3, 4 is 4
2nd smallest of subsequence 3, 4, 2 is 3
2nd smallest of subsequence 4, 2, 1 is 2
2nd smallest of subsequence 2, 1, 5 is 2
Therefore the smallest “K”-value is 2.

Magic Square

Johnny designed a magic square (square of numbers with the same sum for all rows, columns and diagonals i.e. both the **main diagonal** - meaning the diagonal that leads from the top-left corner towards bottom-right corner - and the **antidiagonal** - meaning the diagonal that leads from top-right corner towards bottom-left corner). Write a program to test it.

Task

Write a program that will check if the given square is magic (i.e. has the same sum for all rows, columns and diagonals).

Input

First line: **N** , the size of the square (1 <= N <= 500).
Next N lines: The square, N space separated integers pre line, representing the entries per each row of the square.

Output

First line: **M** , the number of lines that **do not sum up** to the sum of the **main diagonal** (i.e. the one that contains the first element of the square). If the Square is magic, the program should output 0.
Next **M** lines: A sorted (in **incremental order**) list of the lines that do not sum up to the sum of the main diagonal. The rows are numbered 1,2,…,N; the columns are numbered -1,-2,…,-N; and the antidiagonal is numbered zero.

Note: There is a newline character at the end of the last line of the output.

Sample Input 1

```
3
8 1 6
```

3 5 7
4 9 2

Sample Output 1

0

Sample Input 2

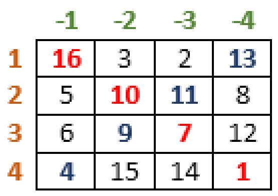
4
16 3 2 13
5 10 11 8
6 9 7 12
4 15 14 1

Sample Output 2

3
-2
-1
0

Explanation of Sample Output 2

The input square looks as follows:



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

The square has 4 rows (labeled from 1 to 4 in orange) and 4 columns (labeled from -1 to -4 in green) as depicted in the image above. The main diagonal and antidiagonal of the square are highlighted in red and blue respectively.

The main diagonal has sum = 16 + 10 + 7 +1 = 34.
The antidiagonal has sum = 13 + 11 + 9 + 4 = 37. This is different to the sum of the main

diagonal so value 0 corresponding to the antidiagonal should be reported.
Row 1 has sum = 16 + 3 + 2 + 13 = 34.
Row 2 has sum = 5 + 10 + 11 + 8 = 34.
Row 3 has sum = 6 + 9 + 7 + 12 = 34.
Row 4 has sum = 4 + 15 + 14 + 1 = 34.
Column -1 has sum = 16 + 5 + 6 + 4 = 31. This is different to the sum of the main diagonal so value -1 should be reported.
Column -2 has sum = 3 + 10 + 9 + 15 = 37. This is different to the sum of the main diagonal so value -2 should be reported.
Column -3 has sum = 2 + 11 + 7 + 14 = 34.
Column -4 has sum = 13 + 8 + 12 + 1 = 34.
Based on the above, there are 3 lines that do not sum up to the sum of the elements of the main diagonal. Since they should be sorted in incremental order, the output should be:

```
3
-2
-1
0
```