The 2nd Universal Cup

Stage 16: Run Twice

December 30-31, 2023

This problem set should contain 11 problems on 24 numbered pages.

Based on

Petrozavodsk Programming Camp

Problem A. Bracket-and-bar Sequences

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

Let us define the set of *regular bracket-and-bar sequences* R recursively. It is the set of strings that can be obtained following only the rules below:

- $\varepsilon \in R$ (empty string)
- $A, B \in R \Rightarrow AB \in R$ (concatenation)
- $A, B \in R \Rightarrow (A|B) \in R$

For example, the sequences containing two triples "(|)" look as folows: "((|)|)", "(|(|))", "(|)(|)".

Establish a correspondence between regular bracket-and-bar sequences of certain length and integers, and implement that correspondence.

Interaction Protocol

In this problem, your solution will be run twice on each test. Each line of input is terminated by an end-of-line character.

First Run

During the first run, the solution encodes bracket-and-bar sequences as integers. The first line contains the word "encode". The second line contains an integer t: the number of test cases ($1 \le t \le 1000$). Each test case is given on two lines: the first line contains an integer n which is the number of "(|)" triples in the sequence ($1 \le n \le 25$), and the second line contains $3n$ characters without spaces, constituting a regular bracket-and-bar sequence with n triples.

Print t lines, one for each test case. On the i-th line, print an integer x_i which you chose to encode the i-th sequence from the input ($0 \le x_i \le 2 \cdot 10^{18}$).

Second Run

During the second run, the solution decodes bracket-and-bar sequences from integers. The first line contains the word "decode". The second line contains an integer t: the number of test cases ($1 \le t \le 1000$). Each test case is given on two lines: the first line contains an integer n which is the number of "(|)" triples in the sequence ($1 \le n \le 25$), and the second line contains the integer printed by your solution for this test case during the first run.

Print t lines, one for each test case. On the i-th line, print the bracket-and-bar sequence from the i-th test case.

Example

For each test, the input during the second run depends on the solution's output during the first run.

Below we show two runs of a certain solution on the first test. It can be seen that this solution encodes the characters by digits 1, 2, and 3, and just prints the resulting string of digits as the encoding integer. Unfortunately, for large n, the strings will become too long.

standard input	*standard output*
encode 3 1 (|) 4 ((((|)|)|)|) 5 (|(|))((|(|))|)	123 111123232323 121233112123323

standard input	*standard output*
decode 3 1 123 4 111123232323 5 121233112123323	(|) ((((|)|)|)|) (|(|))((|(|))|)

Problem B. Even and Odd Combinations

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

Let a *k-combination out of n* be a k-element subset of the n-element set $\{1, 2, \ldots, n\}$. To denote a combination, list its elements in ascending order. For example, 2-combinations out of 3 look as follows: $\{1, 2\}$, $\{1, 3\}$, $\{2, 3\}$.

Let a combination be *even* if the number of its elements is an even number, and *odd* otherwise. For a fixed $n > 0$, consider two sets: A_n, the set of all even combinations out of n, and B_n, the set of all odd combinations out of n. It can be shown that A_n and B_n contain the same number of combinations.

For each $n = 1, 2, \ldots, 50$, your task is as follows. Construct any bijection (a one-to-one correspondence) between the sets A_n and B_n. After that, given an element of one of these sets, print the corresponding element of the other set.

Interaction Protocol

To check that you indeed constructed a bijection, in this problem, your solution will be run twice on each test. Each line of input is terminated by an end-of-line character.

First Run

During the first run, the first line contains an integer t, the number of test cases ($1 \le t \le 1000$). Then follow their descriptions.

Each test case denotes a combination and is given on two input lines. The first of these lines contains two space-separated integers n and k ($1 \le n \le 50$, $0 \le k \le n$). The second one contains k space-separated integers $a_1, a_2, \ldots, a_k$, the elements of the combination ($1 \le a_1 < a_2 < \ldots < a_k \le n$). If $k = 0$, the second line is empty.

For each test case, print the corresponding combination from the other set. Output format for combinations is exactly the same as input format. The printed combination must have the same n as the given one, and k must have different parity. There are no other restrictions on the correspondence.

Second Run

During the second run, the input format is exactly the same as during the first run. However, in each test case, the given combination is not the initial one, but instead the one printed during the first run.

For each test case, as during the first run, print the corresponding combination from the other set. Output format for combinations is exactly the same as input format. As the correspondence must be a bijection, the combination printed during the second run must be the same as the one given during the first run. This is what the jury program will check.

Example

For each test, the input during the second run depends on the solution's output during the first run.

Below we show two runs of a certain solution on the first test.

standard input	standard output
6 3 0 2 1 1 3 3 1 2 3 3 1 1 3 1 2 3 1 3	3 3 1 2 3 2 2 1 2 3 0 3 2 2 3 3 2 1 3 3 2 1 2

standard input	standard output
6 3 3 1 2 3 2 2 1 2 3 0 3 2 2 3 3 2 1 3 3 2 1 2	3 0 2 1 1 3 3 1 2 3 3 1 1 3 1 2 3 1 3

Problem C. Find the Parts

Input file:	*standard input*
Output file:	*standard output*
Time limit:	8 seconds
Memory limit:	512 mebibytes

Two robots, Carl and Clara, are a part of a secret network designed to pass messages.

Robot Clara has got a secret message. The message has a form of a black-and-white rectangle of r rows and c columns which contains $r \times c$ pixels. Each pixel is characterized by brightness: an integer from 0 to 255 (a byte) where 0 is black, 255 is white, and the numbers in between correspond to different shades of gray.

Clara does not know whether the message has some hidden meaning, but it definitely looks like "white noise": each pixel can be considered to have a random value which is independent from other pixels and uniformly distributed among the 256 possible colors.

Clara's job is to answer questions by robot Carl. Each question is formulated as a small black-and-white rectangle. The answer is the coordinates of that rectangle in the original message.

However, before answering questions, Clara has to delete the message. Unfortunately, her memory is limited to a mere 400 kibibytes, so the message may not fit there...

How should Clara act to nevertheless answer all the questions correctly?

Interaction Protocol

In this problem, your solution will be run twice on each test. Each line of input is terminated by an end-of-line character.

In input and output, *bytes* are integers from 0 to 255 inclusive, and they are represented in hexadecimal form: each byte is recorded by exactly two characters, and each of these characters is either a digit `0`–`9` or an uppercase letter `A`–`F`.

First Run

During the first run, the solution gets the message and fills Clara's memory. The first line contains the word "`message`". The second line contains two space-separated integers r and c: the number of rows and columns in the message ($20 \le r, c \le 2000$). Each of the next r lines contains c space-separated bytes: the message itself. The bytes are picked in advance, independently of each other, by a pseudorandom number generator, and all values from the range 0–255 are equally probable.

On the first line, print an integer m, the size of the record in Clara's memory ($0 \le m \le 409\,600$). On the second line, print m space-separated bytes: the contents of that record.

Second Run

During the second run, the solution receives the record in Clara's memory, and then answers Carl's questions. The first line contains the word "`parts`". The second line contains an integer m, the size of the record in Clara's memory ($0 \le m \le 409\,600$). The third line contains m space-separated bytes: the contents of that record. These two lines repeat what the solution printed during the first run.

The next line contains an integer q, the number of Carl's questions ($1 \le q \le 10\,000$). Then the questions follow. Each question starts with a line containing two integers h and w: the number of rows and columns in the question's rectangle ($10 \le h, w \le 20$). Each of the next h lines contains w space-separated bytes: the contents of the rectangle. It is guaranteed that each given rectangle can be uniquely located in the original message. The questions are fixed in advance and do not depend on the results of the first run.

For each question, print a line containing two integers: the row and column in the original message which correspond to the location of the upper left corner of the question's rectangle. The rows are numbered 1 to r from top to bottom, and the columns are numbered 1 to c from left to right.

Example

For each test, the input during the second run depends on the solution's output during the first run.

Below we show two runs of a certain solution on the first test. The memory is shown only partially for brevity. The full version of the example can be seen in `samples.zip`.

standard input

```
message
20 24
33 39 73 4A 5A AA E0 86 96 4B 0B 83 A0 FA 82 9B B0 6E DC 03 1C B9 5B 81
86 3E 23 7B C9 38 77 82 7D 62 EA CE A8 DE 85 6C 36 B3 10 EE 85 6A D5 92
14 BD 58 74 20 7B 36 E1 89 B8 6F 4A F4 8F 17 2E 2F 0F 79 DD AA 9F 6F AD
85 21 B6 2F 58 37 87 7B 3F EE D9 7D 9A E6 AA 12 E0 B6 BB 3D 72 BD 34 A5
E5 8A 73 EE 69 BF E0 0D 5C 57 EF 42 7B 91 07 B8 7D A9 40 0D 4B 52 2D BC
25 F7 4F A7 18 4D 76 EB EB 3E AA 3D C2 19 D3 EE 77 BF C1 38 FF C4 07 C0
CD 2B 79 C3 27 A6 C6 DB D3 17 EA CD 74 BC E5 42 36 F8 D2 86 F9 E9 86 AA
F8 37 39 BF 0C B6 2C 9A F5 04 40 BB D8 FD B4 97 2A 9A A6 D1 9E 2A 60 23
F7 CF 3F 25 CB C1 25 08 0F 1F D2 34 C4 61 27 2E 7B E9 00 FD 86 77 E9 AF
7B 44 57 2E 47 F9 CC A0 03 E3 60 C2 DF C1 F5 6C 59 0E 99 64 3D 7D E7 75
EC C9 BE 91 3B DF 1C DC 61 5C 66 1C B3 26 1C 2E 11 0D 19 BD DC 08 1A 90
BF 93 A0 B9 CD 02 DD E6 49 6F 53 E2 2C 34 10 EA 1A 44 B4 49 7E D5 B6 CB
4A E9 C7 3F F1 FF 24 33 5D 8F D4 26 2E C4 FD 81 FB 96 36 51 F1 38 BE 1E
5A C9 B2 3D 06 99 4F 99 3F 45 DB DA 14 BE 53 D7 B2 2D 64 7B 10 74 0E 70
B6 07 1A B4 F3 25 4D EB 3F 68 72 10 3B 56 F2 A7 C4 A4 28 AE 16 D0 13 CC
91 C4 4D 51 04 39 A8 13 3C 1F 00 57 24 2A FD EA FC EB 77 B8 E1 7D DF 0D
92 51 DA 2A CD A1 F3 97 1A 7A EF 41 DF BD 16 4D 05 4B 78 20 B7 68 38 1C
10 D5 DE 39 58 8F F6 22 8B E8 E8 D0 FB 37 31 33 9E C8 FC 79 62 4F BB 96
5F 04 CB 93 16 9F 15 07 96 27 35 09 AB 79 92 37 44 15 14 A1 4E 04 67 5D
C1 C4 8B 1A 77 E1 D2 4D 06 42 07 A3 1A 67 EC F1 B2 08 96 F6 C3 4E 79 E9
```

standard output

```
484
14 00 18 00 33 39 73 4A 5A AA E0 86 <...> C3 4E 79 E9
```

standard input

```
parts
484
14 00 18 00 33 39 73 4A 5A AA E0 86 <...> C3 4E 79 E9
2
10 10
39 73 4A 5A AA E0 86 96 4B 0B
3E 23 7B C9 38 77 82 7D 62 EA
BD 58 74 20 7B 36 E1 89 B8 6F
21 B6 2F 58 37 87 7B 3F EE D9
8A 73 EE 69 BF E0 0D 5C 57 EF
F7 4F A7 18 4D 76 EB EB 3E AA
2B 79 C3 27 A6 C6 DB D3 17 EA
37 39 BF 0C B6 2C 9A F5 04 40
CF 3F 25 CB C1 25 08 0F 1F D2
44 57 2E 47 F9 CC A0 03 E3 60
11 20
18 4D 76 EB EB 3E AA 3D C2 19 D3 EE 77 BF C1 38 FF C4 07 C0
27 A6 C6 DB D3 17 EA CD 74 BC E5 42 36 F8 D2 86 F9 E9 86 AA
0C B6 2C 9A F5 04 40 BB D8 FD B4 97 2A 9A A6 D1 9E 2A 60 23
CB C1 25 08 0F 1F D2 34 C4 61 27 2E 7B E9 00 FD 86 77 E9 AF
47 F9 CC A0 03 E3 60 C2 DF C1 F5 6C 59 0E 99 64 3D 7D E7 75
3B DF 1C DC 61 5C 66 1C B3 26 1C 2E 11 0D 19 BD DC 08 1A 90
CD 02 DD E6 49 6F 53 E2 2C 34 10 EA 1A 44 B4 49 7E D5 B6 CB
F1 FF 24 33 5D 8F D4 26 2E C4 FD 81 FB 96 36 51 F1 38 BE 1E
06 99 4F 99 3F 45 DB DA 14 BE 53 D7 B2 2D 64 7B 10 74 0E 70
F3 25 4D EB 3F 68 72 10 3B 56 F2 A7 C4 A4 28 AE 16 D0 13 CC
04 39 A8 13 3C 1F 00 57 24 2A FD EA FC EB 77 B8 E1 7D DF 0D
```

standard output

```
1 2
6 5
```

Problem D. Message Made of Noise

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

Alisa wants to send a message to Eva using a *number wire*. The message is one English word.

Unfortunately, right now the number wire transmits just some noise: random integers from 0 to $10^9 - 1$ inclusive. Alisa knows the sequence of the next 10 000 integers that will be transmitted.

Fortunately, Alisa has a superpower: she can erase any number of elements from any positions in the sequence. The relative order of the remaining elements does not change.

Unfortunately, after that, around half of the integers will be lost in transmission: each transmitted integer will disappear with a probability of 1/2. The relative order of the remaining elements once again does not change.

How should Alisa and Eva act to transmit a given word?

Interaction Protocol

In this problem, your solution will be run twice on each test. In input and output, numbers on a single line are separated by spaces. Each line of input is terminated by an end-of-line character.

First Run

During the first run, the solution acts for Alisa. The first line contains the name "`Alisa`". The second line contains one word from English dictionary, its length is from 2 to 15 letters, and it consists of lowercase English letters. The third line contains an integer n, the length of the sequence (in this problem, n is always equal to 10 000). The fourth line contains n integers from 0 to $10^9 - 1$ inclusive: the initial sequence. The numbers are selected in advance by a pseudorandom number generator, all numbers from the range are equiprobable.

The solution should print the numbers that Alisa decided to **leave** in the sequence. On the first line, print an integer m: the number of integers left. On the second line, print the remaining numbers in the order they follow in the initial sequence.

Second Run

During the second run, the solution acts for Eva. The first line contains the name "`Eva`". The second line contains an integer k, the number of remaining integers in the sequence. The third line contains k integers from 0 to $10^9 - 1$ inclusive: the remaining sequence itself. Each number that Alisa decided to leave in the sequence is present with probability 1/2 and missing with probability 1/2. The way the numbers go missing is fixed in advance in each test, so, if solutions make the same choices in the first run, they will get the same sequences for the second run.

Print one English word: the word Alisa should have sent to Eva.

Example

For each test, the input during the second run depends on the solution's output during the first run.

Below we show two runs of a certain solution on the first test. The sequences are shown only partially for brevity. The full version of the example can be seen in `samples.zip`.

standard input
Alisa spark 10000 833080662 16249270 933346436 811379468 <...> 13286897 459644281
standard output
3900 933346436 811379468 877083772 408973036 <...> 583178591 13286897

standard input
Eva 1955 811379468 408973036 585189166 111199534 <...> 226510051 829146141
standard output
spark

Problem E. Four Plus Four

Input file:	*standard input*
Output file:	*standard output*
Time limit:	3 seconds
Memory limit:	512 mebibytes

Queen Marianna has three daughters, the princesses. Marianna keeps her royal seal in a safe. The safe is protected by a *password*: it is an eight-letter English word from the royal dictionary. The password changes every few days.

The Queen regularly goes on vacation, and in the meantime, the princesses learn to reign the kingdom. No one of them knows the password, however, Marianna wants any two princesses to be able to open the safe in case they have an agreement. For that, on three cards, she writes three *keys*: four-letter English words from the royal dictionary. Each key consists **of the password letters**: from the eight letters, four are selected and possibly rearranged, so that the result is a dictionary word. After that, the Queen puts the cards into a hat, and then the three princesses, in sequence, take a random card and keep it to themselves, not showing it to others.

Having the royal dictionary at hand, devise an arrangement for the Queen and the princesses so that, after Marianna hands out the keys, any two princesses could determine the password using the two keys they have.

Interaction Protocol

In this problem, your solution will be run twice on each test. Each line of input is terminated by an end-of-line character. The dictionary is the same in every test and in the example: the words are taken from the freely distributed word list known as ENABLE2K. All words in input and output consist of lowercase English letters.

First Run

During the first run, the solution acts for Queen Marianna. The first line contains the word "`password`". The second line contains an integer n: the number of passwords for which to select the keys ($1 \le n \le 10\,000$). Each of the next n lines contains a single password: an eight-letter word from the royal dictionary. A word can be given as a password only if it is possible to construct **at least three** distinct keys from its letters.

After that, the royal dictionary is given. The description takes four lines and is the same in every test. The first of these lines contains an integer n_8: the number of eight-letter words in the dictionary ($n_8 = 28\,558$). The second line is a space-separated list of the eight-letter words themselves in lexicographical order. The third line contains an integer n_4: the number of four-letter words in the dictionary ($n_4 = 3919$). The fourth line is a space-separated list of the four-letter words themselves in lexicographical order.

Print n lines: for each password, print three keys that Marianna writes on the cards, in any order, separated by spaces. Each key is a four-letter word from the royal dictionary consisting of the password letters: from its eight letters, four are selected and possibly rearranged, so that the result is a dictionary word. There are no other restrictions on key selection: for example, a password could turn into three keys which are all the same, and another could turn into that same key and two different keys.

Second Run

During the second run, the solution acts as the prinecsses. The first line contains the word "`keys`". The second line contains an integer m: the number of pairs of keys ($1 \le m \le 60\,000$). Each of the next m lines contains a pair of keys separated by a space: each such pair is selected by the jury program from some key triple printed during the first run. A pair is selected as follows: first, one key is removed, and second, the remaining two keys may be swapped. The selection is deterministic: if two solutions produced the same output during the first run, they will get the same input during the second run.

After that, the royal dictionary is given. The description takes four lines, and is exactly the same as during the first run.

Print m lines: for each pair of keys, print the right password.

Example

For each test, the input during the second run depends on the solution's output during the first run.

Below we show two runs of a certain solution on the first test. The dictionary is shown only partially for brevity. The full version of the example can be seen in `samples.zip`.

standard input	*standard output*
password 2 password couthier 28558 aardvark aardwolf <...> zyzzyvas 3919 aahs aals abas <...> zori zyme	swap road saws thou thou thou

standard input	*standard output*
keys 4 swap road thou thou saws swap road saws 28558 aardvark aardwolf <...> zyzzyvas 3919 aahs aals abas <...> zori zyme	password couthier password password

Note that the last eight-letter word in the dictionary, "`zyzzyvas`", is an example of a word that can not be given as a password: from its letters, it is only possible to construct one key, "`yays`". Recall that a word can be given as a password only if at least three keys can be constructed from its letters. There are 70 eight-letter words in the dictionary for which this condition does not hold.

Problem F. Mark on a Graph

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

You are given a graph with n vertices and m edges: the graph is undirected and has no self-loops and no multiple edges. You know for a fact that the graph was obtained in one of the two ways.

- The graph is randomly generated: the process starts with a graph with no edges, and then, m times, a random uniformly chosen non-existent edge is added to it.

  In this case, leave a mark on the graph. For that, you can do the following operation from 0 to 5 times: pick a pair of vertices and change the state of the edge between them, adding it if it was not present or removing it otherwise.

- The graph contains a mark, in other words, it is obtained from a random graph by the procedure described above. But after that procedure, the vertices are renumerated randomly, and the edges are given in random order as well. The two vertices of each edge can also be given in any ordder.

  In this case, nothing more has to be done.

Interaction Protocol

In this problem, your solution will be run twice on each test. In input and output, numbers on a single line are separated by spaces. Each line of input is terminated by an end-of-line character.

During each run, the solution gets a graph as input. The first line contains two integers n and m: the number of vertices and edges in the graph. Each of the following m lines contains two integers u and v denoting an edge between vertices u and v in the graph ($1 \le u, v \le n$, $u \ne v$, the bidirectional edges are all distinct).

First Run

During the first run, the given graph is randomly generated in advance according to the problem statement ($n = 1000$, $2000 \le m \le 5000$). On the first line, print the word "`mark`", and on the second line, print the number k of operations with edges ($0 \le k \le 5$). Each of the following k lines must contain two integers u and v denoting the change of state of the edge between vertices u and v ($1 \le u, v \le n$, $u \ne v$).

Second Run

During the second run, the given graph is the one obtained after the first run. However, the vertices are renumerated randomly, the edges are given in random order, and the vertices of each edge are also given in random order. All the shuffles are fixed in advance in each test, so, if solutions make the same choices in the first run, they will get the same inputs for the second run. In this case, print the word "`ok`" on the first line.

Example

For each test, the input during the second run depends on the solution's output during the first run.

Below we show two runs of a certain solution on the first test. The graphs are shown only partially for brevity. The full version of the example can be seen in `samples.zip`.

standard input	*standard output*
1000 3560 603 151 415 20 102 569 895 552 <...> 224 267 651 506	mark 3 763 968 572 286 453 139

standard input	*standard output*
1000 3561 192 768 693 994 786 238 351 329 <...> 100 66 54 819	ok

Problem G. Transfer of Duty

Input file:	*standard input*
Output file:	*standard output*
Time limit:	4 seconds
Memory limit:	512 mebibytes

Anya is going to be an operator at the laboratory today. The operator's desk has one million switches, no kidding! The switches are numbered by integers from 1 to 10^6, and each switch corresponds to a device with the same number. The switches don't show whether the respective devices are on or off, but it is known that toggling a switch changes the state from "on" to "off" and from "off" to "on".

When Anya arrives in the morning, all devices are off. After that, fellow workers come and occasionally toggle the switches.

To optimize energy consumption, after each toggle, the operator has to distinguish between the following classes of states:

- all devices are off,
- exactly one device is on: it is necessary to know which one,
- two or more devices are on.

Sure enough, Anya will be able to do this. But then she will have to transfer the operator duty to her friend Andrei. And during the transfer, she may only leave a short note to him. After reading the note, Andrei will have the exact same task: fellow workers will toggle the switches, and he will have to know the current class of states of the laboratory.

Help the friends devise a way to write the note so that not only Anya, but also Andrei has all the necessary information after each toggle.

Interaction Protocol

In this problem, your solution will be run twice on each test. In each test, all toggles during both runs are fixed in advance. Each line of input is terminated by an end-of-line character.

First Run

During the first run, the solution acts for Anya. The first line contains the word "`start`". The second line contains an integer n, the number of toggles ($1 \le n \le 100\,000$). Each of the following n lines contains one integer: the number of the device (from 1 to 10^6) for which a worker toggled the switch.

For each toggle, print a line with a single integer:

- `0` if all devices are off,
- the number of the device that is on, if there is exactly one such device,
- `-1` if two or more devices are on.

After all the answers, print a single line containing the note Anya will leave to Andrei. The note must have length from 0 to 1000 characters and consist only of characters with ASCII codes from 32 to 126. There are no other restrictions on the note's contents.

Second Run

During the second run, the solution acts for Andrei. The first line contains the word "`resume`". The second line contains the note, exactly as it was printed during the first run. The third line contains an integer m, the number of toggles ($1 \le m \le 100\,000$). Each of the following m lines contains one integer: the number of the device (from 1 to 10^6) for which a worker toggled the switch.

For each toggle, print a line with a single integer, following the same rules as during the first run.

Example

For each test, the input during the second run depends on the solution's output during the first run.

Below we show two runs of a certain solution on the first test.

standard input	*standard output*
start 5 10 14 10 12 10	10 -1 14 -1 -1 3 10 12 14

standard input	*standard output*
resume 3 10 12 14 6 14 277 12 10 277 12	-1 -1 -1 277 0 12

Problem H. Eager Sorting

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

Petya is a sorting robot. In his memory, there is an array of length n (from 1 to 100), and its elements are pairwise distinct integers. The positions in the array are numbered left to right from 1 to n.

Nina is a robot operator. Nina wants to sort this array: make it so that, for each two distinct positions, the number on the left is less than the number on the right. The only available command for that is to compare elements at two positions, i and j. If the element on the left position (it can be position i or j) is greater than the element on the right position, Petya swaps them and displays number `1` on the screen. Otherwise, Petya does nothing with the array and just displays `0` on the screen.

Unfortunately, Petya's power system is damaged, so sometimes the robot shuts down. It looks as follows: when given a command, instead of executing it, Petya just displays `-1` on the screen, and then ignores any subsequent commands.

To make Petya work again, Nina disassembles him and then assembles anew. The array is not altered. Unfortunately, during repairs, Nina forgets which commands she already issued to the robot. After that, Petya works as long as his battery lasts, and then shuts down again.

The battery has enough power for Petya to execute 1500 commands. Petya shuts down exactly twice: after executing x-th command and after executing 1500-th command ($0 < x < 1500$, the value of x is not known to Nina). Knowing all the above, help Nina make it so that, after Petya shuts down for the second time, the array in his memory is sorted.

Interaction Protocol

In this problem, your solution will be run twice on each test. Your solution acts for Nina, and the jury program acts for Petya. Each line of input is terminated by an end-of-line character.

This is an interactive problem. Do not forget to flush the output immediately after printing each command!

Below we describe the interaction of your solution and the jury program: it is the same during both runs.

The first line of input contains an integer n, the size of the array ($1 \le n \le 100$). After that, your solution issues commands, and the jury program executes them and prints the answers.

To issue a command "compare elements at positions i and j", print a line of the form "i j" where $1 \le i, j \le n$. As a result, you will get a line with a single integer:

- `1` means that Petya swapped elements at positions i and j because the left one was greater than the right one;
- `0` means that Petya did not change anything because the left element was not greater than the right one (that is, either the left one was less than the right one, or $i = j$);
- `-1` means that Petya shut down instead of executing the command.

In the latter case, the solution must terminate. In other cases, another command can be issued.

If you want to terminate the interaction before Petya shuts down, instead of a command, print the line "`-1 -1`". After that, the solution must terminate.

In each test, the array before the first run is fixed in advance, and the array before the second run is in the state it was at the end of the first run. Additionally, in each test, the number of commands x after which Petya shuts down for the first time ($0 < x < 1500$) is also fixed in advance. During the second run, the robot will shut down after $1500 - x$ commands, even if the interaction during the first run ended before he shut down for the first time.

Example

Below we show two runs of a certain solution on the first test. Empty lines are added only for readers' convenience: there will be no empty lines during a real run.

standard input	*standard output*	*array*
5		4 2 5 1 3
	1 5	
1		3 2 5 1 4
	1 2	
1		2 3 5 1 4
	2 3	
0		2 3 5 1 4
	3 4	
1		2 3 1 5 4
	4 5	
1		2 3 1 4 5
	2 1	
0		2 3 1 4 5
	3 2	
1		2 1 3 4 5
	4 3	
0		2 1 3 4 5
	1 2	
-1		

standard input	*standard output*	*array*
5		2 1 3 4 5
	1 2	
1		1 2 3 4 5
	2 3	
0		1 2 3 4 5
	1 2	
0		1 2 3 4 5
	-1 -1	

Problem I. Telepathy

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

Brothers Flim and Flam are performing a trick they call "telepathy".

At the beginning, Discord, who is the host, generates two random binary strings a and b. Each string contains $n = 10^6$ digits, and each digit is equal to zero or one equiprobably and independently of other digits. String a is given to Flim, and string b to Flam. Each of them sees only his own string, and doesn't know the string of his brother.

After that, each brother selects $k = 10^5$ distinct positions in the string: not in his own, but in his brother's string that they do not know!

Finally, Discord looks at string a from left to right, and writes down the digits from the positions selected by Flam. Then he looks at string b from left to right and, under the previous line, writes down the digits from the positions selected by Flim. After that, the audience counts how many times a digit from a turned out to be the same as a digit from b written under it. To "prove" that telepathy works, **more than two thirds** of the pairs of digits have to turn out the same, that is, at least 66 667 of them.

Help Flim and Flam to plan how to select positions in each other's strings knowing only their own string, so that they can "prove" that telepathy works.

Consider a small example.

- To keep things short, let $n = 20$ and $k = 5$.
- Let string a be `00101011011110111001`.
- Let string b be `11000111101000011010`.
- Flim sees string a and selects positions $2, 3, 5, 7, 11$ in string b.
- Flam sees string b and selects positions $1, 4, 9, 16, 20$ in string a.
- Discord writes down $a_1 = 0$, $a_4 = 0$, $a_9 = 0$, $a_{16} = 1$, $a_{20} = 1$.
- And under them, $b_2 = 1$, $b_3 = 0$, $b_5 = 0$, $b_7 = 1$, $b_{11} = 1$.
- Out of five pairs, the digits are the same in each pair except the first ($a_1 = 0$ but $b_2 = 1$).
- It means Flim and Flam achieved $4/5$ equalities.
- The portion of equalities is greater than $2/3$, so the brothers "proved" that telepathy works!

Interaction Protocol

In this problem, your solution will be run twice on each test: acting as Flim during the first run and acting as Flam during the second run. The jury program is acting for Discord. Each line of input is terminated by an end-of-line character.

First Run

During the first run, the solution acts for Flim. The first line contains the name "`Flim`". The second line contains two space-separated integers n and k: the length of the string and the number of positions to select (in all tests of the problem, $n = 10^6$ and $k = 10^5$). The third line contains n binary digits without spaces: the string a that is given to Flim.

Print k space-separated integers $1 \le p_1 < p_2 < \ldots < p_k \le n$: the selected positions in string b.

Second Run

During the second run, the solution acts for Flam. The first line contains the name "`Flam`". The second line contains two space-separated integers n and k: the length of the string and the number of positions to select (here, again, $n = 10^6$ and $k = 10^5$). The third line contains n binary digits without spaces: the string b that is given to Flam.

Print k space-separated integers $1 \le q_1 < q_2 < \ldots < q_k \le n$: the selected positions in string a.

After the second run, the jury program compares the digits in k pairs: first a_{q_1} with b_{p_1}, then a_{q_2} with b_{p_2}, and so on. If the equality holds in more than $\frac{2}{3}k$ of these pairs, the solution gets `OK` for the test. Otherwise, the outcome is `Wrong Answer`.

In this problem, there are one downloadable example and 50 secret tests. All binary strings are generated in advance by a pseudorandom number generator: each digit of each string is either zero or one, equiprobably and independently of other digits.

Example

Below we show two runs of a certain solution on the first test. The strings and answers are shown only partially for brevity. The full version of the example can be seen in `samples.zip`. There are 66 859 equal pairs in the example.

standard input	*standard output*
Flim 1000000 100000 110111111110<...>11010111	3 14 25 <...> 999979 999990

standard input	*standard output*
Flam 1000000 100000 000000110100<...>10011111	7 16 21 <...> 999977 999992

Problem J. Tetra-puzzle

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

Tetra-puzzle is a turn-based game for one player resembling Tetris. The game goes on a square board of size 5×5 squares. Initially, the board is empty.

On each turn, the player has to place a tetramino of the given kind on the board. A tetramino is a side-connected piece consisting of four squares. Each piece can be rotated, flipped and translated, and has to be placed so that it occupies four empty squares of the board. After that, take all rows and columns of the board where all squares are filled, and simultaneously clear them up: every square of these rows and columns becomes empty again. The player loses if they can not make a move.

There are five kinds of tetramino pieces in total. Each kind is denoted by its name, an uppercase English letter resembling the form of the piece:

```
.....   .....   .....   .....   .....
.*...   .*...   .**..   .***.   .**..
.*...   .*...   .**..   ..*..   ..**.
.*...   .**..   .....   .....   .....
.*...   .....   .....   .....   .....

  I       L       O       T       Z
```

The game has two modes: basic mode and preparation mode. In basic mode, the player gets n pieces one by one, and has to place the given piece before getting the piece for the next move. The goal is to make all n moves successfully.

In preparation mode, the player plans their next game in basic mode. For preparation, the player gets n randomly generated pairs of pieces right away: the pairs for first, second, ..., n-th move. From each pair, the player has to select one piece, and that piece is what the player will get for the respective move in basic mode.

Your task is to successfully complete the game, first in preparation mode and then in basic mode using the prepared pieces.

Interaction Protocol

In this problem, your solution will be run twice on each test. Each line of input is terminated by an end-of-line character.

During the second run, this is an interactive problem. Do not forget to flush the output immediately after printing each move!

First Run

During the first run, you play in preparation mode. The first line contains the word "`prepare`". The second line contains an integer n, the number of turns ($1 \le n \le 1000$). The third line contains n space-separated pairs of pieces: two choices for first, second, ..., n-th move. Each pair is given by two uppercase letters from the set "`ILOTZ`" corresponding to the kinds of pieces in the pair. The letters in a pair are different and can be given in any order. In each test, each pair is selected randomly in advance, equiprobably from all possible choices and independently from other pairs.

Print a line containing n letters without spaces: for each move, print which one of the two pieces you choose.

Second Run

During the second run, you play in basic mode. The first line contains the word "`play`". The second line

contains an integer n, the number of turns which is the same as during the first run ($1 \le n \le 1000$). Then follow n turns.

On each turn, firstly, the solution reads a line with a single letter: the kind of tetramino selected for this turn during the first run. In response, print the board with the given piece already placed on it, but before clearing up the filled rows and columns. The board takes of 5 lines each of which consists of 5 characters. Character "`.`" (dot, ASCII code 46) means an empty square, character "`#`" (hash, ASCII code 35) is a square occupied on one of the previous turns, and character "`*`" (asterisk, ASCII code 42) is a square of the latest piece put on the board.

If the above formatting rules are satisfied, but the move is invalid, the solution will get "`Wrong Answer`" for the test.

If the move is valid, all filled rows and columns are cleared up, and the occupied squares must be denoted by "`#`" character further on. After that comes the next turn.

If all n turns already happened, the solution will get "`OK`" for the test.

Example

For each test, the input during the second run depends on the solution's output during the first run.

Below we show two runs of a certain solution on the first test. Empty lines are added only for readers' convenience: there will be no empty lines during a real run.

standard input	*standard output*
prepare 6 TO LO ZI LI LT LT	OLZITT

standard input	*standard output*
play 6 O	..... **... **... ....
L	..*** ##..* ##... ....
Z	..### ##**# ##.** ....
I	..### ..*.. ##*## ..*.. ..*..
T	***## .*... ....
T	..... *#... **... *.... .....

Problem K. Trijection

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

There are many types of combinatorial objects such that the number of distinct objects of a given size n is a Catalan number $C_n = \frac{(2n)!}{n!(n+1)!}$. Here are the first few Catalan numbers: $C_0 = 1$, $C_1 = 1$, $C_2 = 2$, $C_3 = 5$, $C_4 = 14$, $C_5 = 42$, ... Consider three such types of objects:

- Skew polyominoes with perimeter $2n+2$. These are collections of squares on a rectangular board of $h \times w$ squares where $h+w = n+1$. A polyomino must be side-connected. The bottom left square and the top right square are occupied by the polyomino. Other squares can be either free or occupied, but the following conditions must hold:
  - in each row and in each column, the occupied squares form a continuous segment;
  - each column segment starts higher or at the same level as the start of its left neighbor;
  - each column segment ends higher or at the same level as the end of its left neighbor;
  - each row segment starts to the right or at the same point as the start of its lower neighbor.
  - each row segment ends to the right or at the same point as the end of its lower neighbor.

  Here are all skew polyominoes of size $n = 4$:

```
poly      poly      poly      poly      poly      poly      poly
4 1       3 2       3 2       3 2       3 2       3 2       3 2
#         .#        .#        ##        ##        ##        .#
#         .#        ##        #.        ##        ##        ##
#         ##        #.        #.        #.        ##        ##
#
```

```
          poly      poly      poly      poly      poly      poly
poly      2 3       2 3       2 3       2 3       2 3       2 3
1 4       ..#       .##       ###       ###       ###       .##
####      ###       ##.       ##.       #..       ###       ###
```

- 321-avoiding permutations of length n. These are permutations $p_1, p_2, \ldots, p_n$ of elements $1, 2, \ldots, n$ which don't contain a triple of positions $i < j < k$ such that $p_i > p_j > p_k$.

  Here are all 321-avoiding permutations of size $n = 4$:

```
perm      perm      perm      perm      perm      perm      perm
1 2 3 4   1 3 2 4   1 4 2 3   2 1 4 3   2 3 4 1   3 1 2 4   3 4 1 2

perm      perm      perm      perm      perm      perm      perm
1 2 4 3   1 3 4 2   2 1 3 4   2 3 1 4   2 4 1 3   3 1 4 2   4 1 2 3
```

- Triangulations of a convex $(n+2)$-gon. Label the vertices of the polygon by integers from 1 to $n+2$ in the order of traversal. In each of the n triangles, arrange the vertex numbers in ascending order. Next, arrange the triangles themselves in ascending order as triples of integers.

  Here are all triangulations of size $n = 4$:

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

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

Let us fix the number n and consider three sets:

- A_n, the set of skew polyominoes of size n,
- B_n, the set of 321-avoiding permutations of size n,
- C_n, the set of triangulations of size n.

Construct a *trijection* between these sets. A trijection is like a bijection, but there are three sets instead of two. Formally, invent a function $f_n : A_n \cup B_n \cup C_n \to A_n \cup B_n \cup C_n$ such that its value for each object of any type is necessarily an object of a **different** type, and additionally, $f_n(f_n(x)) = x$ for every x (in other words, f_n is an inverse to itself).

After that, for given objects of the three types, print the results of trijection.

Additional constraint: the given number n **can not** have the form $2^k - 1$.

Interaction Protocol

In this problem, your solution will be run twice on each test. Each line of input is terminated by an end-of-line character.

First Run

During the first run, the first line contains two space-separated integers n and t: the size, which is the same for all objects, and the number of objects ($2 \le n \le 35$, $1 \le t \le 1000$, the number n can not have the form $2^k - 1$). Then t objects are given. Each object description starts with a line indicating its type, followed by one or more lines describing the object itself, which depend on the type. The detailed desription of types and the formatting rules for objects are shown in the statement above and in the example below.

On the first line, print n and t separated by a space (this line is part of the format for convenience, so that it is possible to use solution's output as input without changes). After that, print t objects: the results of the trijection for the t given objects. The output format for the objects is the same as the input format.

Second Run

During the second run, the input and output formats are the same as during the first run. However, instead of the t initial objects, the objects given in the input are the ones printed during the first run, **reordered randomly**.

The t initial objects are fixed in advance in each test. The random permutation applied between the first and the second run is also fixed in advance.

Example

For each test, the input during the second run depends on the solution's output during the first run.

Below we show two runs of a certain solution on the first test. Note how the output of the second run is reordered input of the first run.

standard input	*standard output*
5 4 poly 4 2 .# ## ## #. perm 4 1 5 2 3 triang 1 2 4 1 4 5 1 5 7 2 3 4 5 6 7 perm 2 1 3 5 4	5 4 perm 3 1 4 2 5 poly 4 2 ## ## ## #. poly 3 3 .## ### ##. triang 1 2 3 1 3 7 3 4 7 4 5 7 5 6 7

standard input	*standard output*
5 4 poly 4 2 ## ## ## #. triang 1 2 3 1 3 7 3 4 7 4 5 7 5 6 7 poly 3 3 .## ### ##. perm 3 1 4 2 5	5 4 perm 4 1 5 2 3 perm 2 1 3 5 4 triang 1 2 4 1 4 5 1 5 7 2 3 4 5 6 7 poly 4 2 .# ## ## #.