The 4th Universal Cup

Stage 17: Grand Prix of St. Petersburg

February 21-22, 2026

This problem set should contain 13 problems on 28 numbered pages.

Based on

Petrozavodsk Programming Camp

Prepared by

St. Petersburg State University

Problem A. Anthill/Honeypot Simulator

Time limit:	2 seconds
Memory limit:	1024 mebibytes

An anthill with $N \approx 10^{100}$ ants is located on an infinite plane. Apart from that, the plane contains only a circular honeypot of radius R with center at a distance d from the anthill.

All ants are originally located in the anthill. Then all of them start moving independently of each other. Each ant randomly selects the direction of its movement and moves in the chosen direction with speed $(\Delta t)^{-1/2}$ for Δt units of time, thus traveling for a distance of $\sqrt{\Delta t}$ units, where $\Delta t \approx 10^{-100}$ is a very small number. Then each ant repeats the whole process again and again. Whenever an ant is located inside the honeypot, it collects one unit of honey per each unit of time spent inside.

Your task is to simulate the anthill and the honeypot for T units of time after the start and compute two numbers: the fraction f of ants that end up inside the honeypot after T units of time and the average amount h of honey collected by each ant.

Input

The first line of input contains a single integer k, the number of test cases ($1 \le k \le 1000$).

Each of the next k lines describes a test case and contains three real numbers, d, R, and T, given with at most four digits after the decimal point ($0 \le d \le 100$; $0.01 \le R \le 100$; $0.01 \le T \le 100$).

Output

You have to output k lines, one line for each test case. Each line should contain the values of f and h, separated by one or several spaces. You have to output a value of f that differs from the correct value by at most 10^{-8}, and a value of h that differs from the correct value by at most $10^{-8} \cdot T$.

Example

standard input	*standard output*
7 0 1 1 2 1 3 1 2 0.999 1 2 1 1 2 1.001 1 1 1 0 10 10	0.6321205588285579 0.8515044932240781 0.09157635496981011 0.1733100735950181 0.8767916692090618 0.9555113318223997 0.8766185521451777 0.9563880369284812 0.8764454902564369 0.9572645689450815 0.3457458387231645 0.4012213611982974 0.9999546000702375 9.999961697595344

Problem B. Missing Number

Time limit:	2 seconds
Memory limit:	1024 mebibytes

You are given an array of n integers from 1 to $n+1$. It is possible that some of the integers appear multiple times in the array. Still, it is clear that at least one of the integers from 1 to $n+1$ is missing from the array. Your task is to find *any* such integer. Normally, it would be easy, but there is an unfortunate issue: there is not enough memory on your computer to represent the whole array.

Hence, you are asked to produce a *streaming* algorithm instead. Informally, the elements of the array are given to you one-by-one with no way to randomly access them. However, there is one solace: after receiving the last element of the array, you can restart reading it from the beginning. Ideally, you would like to avoid such a situation and solve the whole problem in one pass. However, sometimes (including this very problem) multiple passes over the input may be necessary to avoid using too much memory. You need to balance both concerns and *simultaneously* use not too much memory and make not too many passes over the array (the problem is trivial if you are allowed to store n bits of memory or use n passes).

There is still one catch, however. For technical reasons, you need to produce a *deterministic finite automaton* (DFA) over the alphabet $\{0, 1, \ldots, n+1\}$ implementing the logic of your algorithm. The characters of the alphabet correspond to the information you receive when reading the stream: numbers from 1 to $n+1$ represent reading a corresponding integer from the array, while the number 0 represents the fact that you have reached the end of the array. After the first 7 times you have reached the end of the array, the array will be given to you again in the same order as before, always ending with a 0 character representing the end of the array. Hence, your DFA will be fed the string $a_1, a_2, \ldots, a_n, 0$ a total of 8 times (here, $a_1, a_2, \ldots, a_n$ is the hidden array). Finally, at any moment of the execution, your automaton can "say" that it already knows some correct answer to the problem; in that case, the execution is halted. If your DFA has not done so by the time it has read the whole stream (including the 0 character at the end) 8 times, it is not accepted as a solution. Similarly, if it halts, but with an incorrect answer, it is not accepted as a solution either.

Let us state the problem formally. There is a hidden array $a_1, a_2, \ldots, a_n$ of n integers from 1 to $n+1$. In the example, $n = 4$, in all other tests $n = 250$. You need to specify a finite state machine with at most 4000 states, numbered from 0 to $q-1$, where $q \le 4000$ is the number of states. For the state i ($0 \le i \le q-1$), you need to specify $n+2$ numbers $\delta(i, 0)$, $\delta(i, 1)$, ..., $\delta(i, n+1)$. The number $\delta(i, j)$ represents the behavior of your DFA when it is in the state i and reads the number j from the input. It must be an integer from $-(n+1)$ to $q-1$ inclusive. Negative integers correspond to the fact that the execution is halted and an answer is given; specifically, $-s$ corresponds to your DFA claiming that $+s$ is a correct answer to the problem. Nonnegative integers correspond to the fact that the execution is continued and the state of the automaton changes from i to $\delta(i, j)$. The execution of the automaton starts in the state 0.

Your DFA is considered to work correctly on the array $a_1, a_2, \ldots, a_n$ if it halts with a correct answer at some point of reading the string $a_1, a_2, \ldots, a_n, 0, a_1, a_2, \ldots, a_n, 0, \ldots, a_1, a_2, \ldots, a_n, 0$, where the array is repeated 8 times. Your DFA is considered to work correctly in general if it works correctly on all possible arrays $a_1, a_2, \ldots, a_n$ of integers from 1 to $n+1$ (hence, probabilistic solutions are discouraged).

For $n = 4$, we will check your DFA on all possible arrays. For $n = 250$, we do not have means to check it on all arrays, so we will check it on less than 5000 specially selected ones. There are two tests in the problem; both tests effectively check that your approach works correctly on many arrays simultaneously.

Input

One integer n, which is either equal to 4 or to 250.

Output

On the first line, print a single integer q ($1 \le q \le 4000$) denoting the number of states in your DFA.

On the i-th of the following q lines, print $n+2$ integers $\delta(i-1,0)$, $\delta(i-1,1)$, ..., $\delta(i-1,n+1)$. They should all be from $-(n+1)$ to $q-1$ inclusive and correspond to the transitions of the automaton. Note that **all** numbers should be in this range: even if some combination of the state i and the input number j can never be reached on any valid input, the value of $\delta(i,j)$ that you specify should still be in the range $[-(n+1), q-1]$.

Example

standard input	*standard output*
4	8 -3 0 0 1 0 0 2 1 1 1 1 1 -1 3 2 2 2 2 4 3 3 3 3 3 -5 4 4 4 4 5 6 5 5 5 5 5 -2 6 7 6 6 6 -4 7 7 7 7 7

Note

The answer for the example is a very boring automaton that uses four passes to check whether the array includes numbers 3, 1, 5, and 2 (in that order). Each of the passes effectively uses only two states that correspond to whether it has already found the number it is looking for in the current pass. If it reaches the end of the array (the character 0) without finding the necessary number, it immediately halts, saying that the necessary number is indeed missing. Otherwise, it proceeds to use the next pass to search for the next number. Finally, if it has found all four necessary integers, then the number 4 has to be missing. Hence, it does not do a fifth pass looking for the number 4 (but it could have without violating the restrictions on the numbers of the automaton states and used passes).

Of course, this naive approach has no chance of working for $n = 250$, and something much better is needed. The main reason for the example's existence is to illustrate the output format.

Problem C. String Workshop

Time limit:	5.5 seconds
Memory limit:	1024 mebibytes

Cleopatra loves strings. When she is given a string s of length n (consisting of characters $s[1], \ldots, s[n]$), she immediately starts sorting it. During the sorting process, she performs a sequence of swaps of the form $s[i] \leftrightarrow s[i+1]$, where $i \in \{1, 2, \ldots, n-1\}$. In the end, she returns the sorted string and demands a number of gold coins equal to the sum of i for all the swaps made (if a swap was made multiple times with the same i, then that i will be counted in the sum as many times as it was swapped). Cleopatra is not wasteful: among all the ways to sort the string, she chooses the one where the number of coins she receives for sorting is minimal.

Catherine de' Medici also loves strings. She has a string of length n, and she wants to play with it. Specifically, she is interested in the following operations:

1. "`1` i c" — replace $s[i]$ with c;
2. "`2` i j" — take a copy of the substring s from the i-th to the j-th character inclusive ($i \le j$) and sort it, while the characters of the string s remain unchanged.
3. "`3` i j" — extract the substring s from the i-th to the j-th character inclusive ($i \le j$), sort it, and insert it back into the string s at the same position.

Catherine de' Medici can handle the first type of queries herself, but the second and third types of queries are too complex, and she will go to Cleopatra for them. When assessing the cost of this procedure, Cleopatra will sum not the indices from the string s (from i to j), but the indices relative to the beginning of the substring (from 1 to $j - i + 1$).

Catherine de' Medici is also not wasteful and carefully monitors her budget. Therefore, for each of her requests to Cleopatra, calculate how many gold coins Catherine de' Medici will have to spend.

Input

The first line of input contains an integer t — the number of test cases ($1 \le t \le 3 \cdot 10^5$). For each test case:

The first line contains two integers, n and q — the length of the string and the number of queries ($1 \le n, q \le 3 \cdot 10^5$). The next line contains a string of n uppercase English letters. The following q lines describe the queries. Each query description has one of the three types:

1. "`1` i c" — replace $s[i]$ with c ($1 \le i \le n$; $c \in \{\texttt{A}, \texttt{B}, \ldots, \texttt{Z}\}$);
2. "`2` i j" — find the cost of sorting the substring $s[i..j]$, while the string s remains unchanged ($1 \le i, j \le n$);
3. "`3` i j" — find the cost of sorting the substring $s[i..j]$, while the substring $s[i..j]$ in the string s is replaced with the sorted one ($1 \le i, j \le n$).

In each test case, there will be at least one query of the second or third type.

The sum of n over all test cases is at most $3 \cdot 10^5$. The sum of q over all test cases is at most $3 \cdot 10^5$.

Output

For each test case, output in a separate line the answers to all queries of the second and third type, that is, the number of coins that Cleopatra will receive for each of the sorts.

Examples

standard input	*standard output*
1 5 20 DBCAA 2 1 5 2 2 4 1 3 Z 2 1 3 3 1 5 2 1 5 1 5 A 3 2 4 2 1 5 1 1 C 2 1 2 3 4 5 2 3 5 1 2 B 2 1 5 3 1 1 2 1 1 1 4 D 2 1 5 3 1 5	18 3 1 16 0 0 7 1 1 1 9 0 0 3 3
3 3 5 CBA 2 1 3 3 1 3 2 1 3 1 2 C 2 1 2 5 5 ABCDE 2 2 5 1 5 A 2 1 5 3 1 5 2 1 5 6 5 YYYYYY 2 1 6 1 3 A 2 1 6 3 2 5 2 1 6	4 4 0 0 0 9 9 0 0 3 1 1

Note

Consider the first example. In it, the initial string is "DBCAA". Let's go through all 20 queries in order:

1. Query "2 1 5". Current string: "DBCAA". Substring $t[1..5]$ = "DBCAA", answer: 18. The string does not change.
2. Query "2 2 4". String: "DBCAA". Substring $t[2..4]$ = "BCA", answer: 3. The string does not change.
3. Query "1 3 Z". Update: $t[3] \leftarrow$ "Z". Was "DBCAA", now "DBZAA".
4. Query "2 1 3". String: "DBZAA". Substring $t[1..3]$ = "DBZ", answer: 1. The string does not change.
5. Query "3 1 5". String: "DBZAA". Substring $t[1..5]$ = "DBZAA", answer: 16. After mutation, sorting the substring: "DBZAA" $\rightarrow$ "AABDZ". The string became "AABDZ".
6. Query "2 1 5". String: "AABDZ". Substring "AABDZ", answer: 0. The string does not change.
7. Query "1 5 A". Was "AABDZ", now "AABDA".
8. Query "3 2 4". String: "AABDA". Substring $t[2..4]$ = "ABD", answer: 0. After sorting "ABD" does not change, the string remains "AABDA".
9. Query "2 1 5". String: "AABDA". Substring "AABDA", answer: 7. The string does not change.
10. Query "1 1 C". Was "AABDA", now "CABDA".
11. Query "2 1 2". String: "CABDA". Substring $t[1..2]$ = "CA", answer: 1. The string does not change.
12. Query "3 4 5". String: "CABDA". Substring $t[4..5]$ = "DA", answer: 1. After mutation: "DA" $\rightarrow$ "AD". The string became "CABAD".
13. Query "2 3 5". String: "CABAD". Substring $t[3..5]$ = "BAD", answer: 1. The string does not change.
14. Query "1 2 B". Was "CABAD", now "CBBAD".
15. Query "2 1 5". String: "CBBAD". Substring "CBBAD", answer: 9. The string does not change.
16. Query "3 1 1". String: "CBBAD". Substring $t[1..1]$ = "C", answer: 0. Sorting a length of 1 does not change anything, the string remains "CBBAD".
17. Query "2 1 1". String: "CBBAD". Substring $t[1..1]$ = "C", answer: 0. The string does not change.
18. Query "1 4 D". Was "CBBAD", now "CBBDD".
19. Query "2 1 5". String: "CBBDD". Substring $t[1..5]$ = "CBBDD", answer: 3. The string does not change.
20. Query "3 1 5". String: "CBBDD". Substring $t[1..5]$ = "CBBDD", answer: 3. After mutation: "CBBDD" $\rightarrow$ "BBCDD". The final string is "BBCDD".

This page is intentionally left blank

Problem D. Distinguishable Distributions

Time limit:	1 second
Memory limit:	1024 mebibytes

This is an interactive problem.

Lazy Sam and diligent Sarah have been tasked with coming up with two interesting distributions over the set of strings of length n over the alphabet $\{\texttt{H}, \texttt{T}\}$. Lazy Sam will simply flip a coin n times (with each side showing up with a probability of $\frac{1}{2}$; all coin flips are independent) and write down "H" for each head and "T" for each tail. You want to help diligent Sarah teach lazy Sam a lesson. You are given an integer t. Help Sarah come up with a distribution that is t-close to the uniform distribution, while interactively proving that you can distinguish between the distributions of lazy Sam and diligent Sarah with oracle access of length t.

Formally, for some integer n, consider the set $S = \{\texttt{H}, \texttt{T}\}^n$ of size 2^n, consisting of all possible strings of length n over the alphabet of two characters: "H" and "T". A *distribution* over the set S is defined as an array of non-negative real numbers $\{a_s\}_{s \in S}$ of length 2^n indexed by the elements of S, such that their sum equals 1. One of the simplest distributions is the *uniform* distribution, where all a_s are equal to 2^{-n}.

For a pair of numbers $\varepsilon \in [0; +\infty)$ and $k \in \{0, 1, \ldots, n\}$, we say that $\{a_s\}_{s \in S}$ is an (ε, k)-*independent* distribution if, for any choice of k indices $1 \le i_1 < i_2 < \ldots < i_k \le n$ and any subset $T \subseteq \{\texttt{H}, \texttt{T}\}^k$ of strings of length k over the alphabet $\{\texttt{H}, \texttt{T}\}$, the following holds: the sum of a_s over the $2^{n-k}|T|$ strings s whose subsequence $s[i_1]s[i_2]\ldots s[i_k]$ belongs to T differs from $\frac{|T|}{2^k}$ by no more than ε. In probabilistic terms, this can be expressed as:

$$\forall 1 \le i_1 < i_2 < \ldots < i_k \le n \quad \forall T \subseteq \{\texttt{H}, \texttt{T}\}^k \quad \frac{|T|}{2^k} - \varepsilon \le \mathbb{P}_{s \leftarrow \{a_s\}_{s \in S}} \{s[i_1]s[i_2]\ldots s[i_k] \in T\} \le \frac{|T|}{2^k} + \varepsilon.$$

Intuitively, this means that to distinguish the distribution from uniform, you will either need to distinguish probabilities whose difference does not exceed ε, or look at more than k bits of the strings produced by the distribution.

The distribution $\{a_s\}_{s \in S}$ is called t-*close* to the uniform distribution if it is $(2^{-t}t, t)$-independent.

In the definition of (ε, k)-independence, it was implied that you must first choose k indices and only then study the probability distribution over those k indices. In the oracle access model of length k, everything is different. A random string of length n is generated, and then k rounds of the following format are conducted: you call an integer $i \in \{1, 2, \ldots, n\}$, and in response, you will be told $s[i]$. You can decide which index to call next, taking into account all previously asked questions and their answers.

To prove that you can distinguish the distributions of lazy Sam and diligent Sarah, you will need to, having oracle access of length t to m strings of length n sampled from their distributions (each independently and uniformly generated from either lazy Sam's or diligent Sarah's distribution), conclude for each of them from whose distribution it was taken, and correctly guess at least $0.75m - 2\sqrt{m}$ times.

Interaction Protocol

The first line contains an integer t, denoting the required closeness to a uniform distribution and the length of oracle access ($1 \le t \le 5$).

In response, print a line with a single integer n, denoting the length of the strings for which you are ready to provide the distribution and an interactive proof of distinguishability ($t \le n \le 20$).

In the next line, print a line with 2^n integers b_s, defining your distribution ($0 \le b_s < 2^{64}$; $\sum_{s \in S} b_s > 0$). Based on this data, the jury will choose the distribution $\{a_s\}_{s \in S}$ according to the rule $a_s = b_s / \sum_{s \in S} b_s$. The values of b_s must be listed in lexicographical order of the strings s: for example, for $n = 3$, the numbers should be printed in the following order: $b_{\texttt{HHH}}$, $b_{\texttt{HHT}}$, $b_{\texttt{HTH}}$, $b_{\texttt{HTT}}$, $b_{\texttt{THH}}$, $b_{\texttt{THT}}$, $b_{\texttt{TTH}}$, $b_{\texttt{TTT}}$.

Next, read an integer m, denoting the number of interactive checks ($1 \le m \le 10^4$). Following this, the m

checks will be presented, which need to be completed one after another.

At the beginning of each check, the jury will first choose the distribution $\{a_s\}_{s \in S}$ either from lazy Sam or diligent Sarah with a probability of $\frac{1}{2}$. After that, the jury samples a random string $s \sim \{a_s\}_{s \in S}$: any string s can be obtained with probability a_s. The jury then gives you the opportunity to ask no more than t questions about the characters of the string s. To ask a question, print a line with an integer $q \in \{1, 2, \ldots, n\}$, denoting the index of the character of interest. After that, read the character "`H`" or "`T`" which is the answer to the query. After asking from zero to t questions, print a line with the name "`Sam`" if you believe that s was taken from lazy Sam's distribution, or "`Sarah`" if you believe that s was taken from diligent Sarah's distribution. After this, immediately move on to the next check until you have completed all m.

Strings can be printed in any case.

After each printed line, do not forget to flush the output buffer.

Example

standard input	*standard output*
2 3 T T T T T 	 3 7 4 8 9 7 6 7 1 1 2 SAM 1 2 Sarah 3 sARaH

Problem E. Four Sages Around an Oak Tree

Time limit:	1.5 seconds
Memory limit:	1024 mebibytes

This is a run-twice problem; both runs are interactive.

Around a wide oak tree stand four sages. Each sage is wearing a red, green, or blue hat on their head. Each sage can see the colors of the hats of both their neighbors but cannot see their own hat color (the hat does not hang over their eyes) or the color of the hat of the sage opposite them (which is blocked by the wide oak tree). Each sage must, without communicating with the others, write down the color of their hat on a piece of paper. Develop a protocol such that at least one of the four sages will definitely guess the color of their hat.

Interaction Protocol

The first line contains two integers t and s, denoting the number of test cases and the run number of the solution ($1 \le t \le 100$; $s \in \{1, 2\}$). Following this are descriptions of t test cases, each on a separate line.

If $s = 1$, the test case consists of four words, c_1, c_2, c_3, and c_4, denoting the colors of the hats of the participants ($c_i \in \{\texttt{red}, \texttt{green}, \texttt{blue}\}$). In response, output a line with a single integer m ($1 \le m \le 4$), denoting the number of the sage who, according to your protocol, will guess the color of their hat (if there are multiple, output any).

If $s = 2$, the test case consists of an integer m and two words, ℓ and r, denoting the number of the sage who must guess the color of their hat, and the colors of their left and right neighbors' hats ($1 \le m \le 4$; $\ell, r \in \{\texttt{red}, \texttt{green}, \texttt{blue}\}$; $\ell = c_{(m+2) \bmod 4+1}$; $r = c_{m \bmod 4+1}$). In response, output a line with a word g that is the color of the hat of the m-th sage. The answer will be accepted if $g = c_m$. The interactor is case-insensitive, so you can print each letter in any case (uppercase or lowercase).

To prevent passing information between test cases, the jury has taken the following measures:

- Both runs are interactive. Thus, you will not receive the next test case until you output the answer to all previous ones. After printing each line, do not forget to flush the output stream (for example, by using `fflush(stdout)` or `cout.flush()` in C++, `sys.stdout.flush()` in Python, or `System.out.flush()` in Java or Kotlin). Otherwise, you will likely receive an **Idleness Limit Exceeded** error.
- During the second run, the jury may rearrange the t test cases in any order, which may not match the order in which they appeared in the first run.
- During the second run, the jury may also add counterfeit test cases that do not correspond to any test case from the first run. For these cases, any answer of "`red`", "`green`", or "`blue`" will be considered correct; they are only needed so that your program cannot easily distinguish a real test case from a counterfeit one and pass information between cases. Thus, t in the second run may exceed t in the first run. The total number of real and counterfeit cases cannot exceed 100.

Examples

standard input	*standard output*
1 1 green red blue red	2
4 2 2 blue green 4 red blue 3 red red 4 red blue	blue BLUE blUE Blue

Problem F. Pluses and Minuses

Time limit: 3 seconds
Memory limit: 1024 mebibytes

Misha wrote down a few integers that he considers beautiful in a notebook and went to play with pluses and minuses. He considers a string of pluses and minuses to be *balanced* if it has an equal number of both signs and ends with a beautiful number of minuses. How many balanced strings of length $2, 4, \ldots, 2n$ are there? The answers may be very large, so find them modulo 998 244 353.

Input

The first line contains two integers, n and m: the maximum length of a string and the number of integers in Misha's notebook ($1 \le n, m \le 200\,000$).

The next line contains m distinct non-negative integers in the range $[0, n]$: the numbers from Misha's notebook.

Output

Output n lines with answers modulo 998 244 353.

Example

standard input	*standard output*
5 3 1 2 3	1 3 10 34 120

This page is intentionally left blank

Problem G. Traffic Lights

Time limit:	5.5 seconds
Memory limit:	1024 mebibytes

You are given a tree where each vertex has a light bulb that shines red, yellow, or green. Answer many queries of the following types:

- "`1` v c" — change the color of the light bulb at vertex v to color c;
- "`2` v" — given vertex v, which is guaranteed to shine yellow, find the shortest *traffic light* in the tree that passes through v, and output the number of edges in it. A traffic light is defined as a simple path in the tree, one end of which is red, the other is green, and at least one of the internal vertices of the path is yellow.

Input

The first line of input contains an integer t, the number of test cases ($1 \le t \le 2 \cdot 10^5$). For each test case:

The first line contains two integers, n and q — the number of vertices in the tree and the number of queries ($1 \le n, q \le 2 \cdot 10^5$).

Each of the next $n - 1$ lines contains a pair of integers, u_i and v_i — the ends of an edge of the tree ($1 \le u_i, v_i \le n$; $u_i \ne v_i$). It is guaranteed that these edges form a tree.

The following line contains a string of length n consisting of the letters "`R`", "`Y`", "`G`", where the i-th letter denotes the initial color of the light bulb at the i-th vertex. "`R`" means red light, "`Y`" — yellow, "`G`" — green.

Then follow the descriptions of q queries, one per line:

- "`1` v c" — change the color of the light bulb at vertex v to color c ($1 \le v \le n$; $c \in \{\texttt{R}, \texttt{Y}, \texttt{G}\}$);
- "`2` v" — output the length of the shortest traffic light passing through v ($1 \le v \le n$). It is guaranteed that at the time of this query, the light bulb at vertex v shines yellow. It is guaranteed that there will be at least one query of this type.

It is guaranteed that the sum of n across all test cases does not exceed $2 \cdot 10^5$. It is guaranteed that the sum of q across all test cases does not exceed $2 \cdot 10^5$.

Output

For each test case and for each query of the type "`2` v", output a single number d — the number of edges on the shortest simple path in the tree that starts at a red vertex, passes through the yellow vertex v, and ends at a green vertex. If such paths do not exist, output -1.

Examples

standard input	*standard output*
1 7 7 1 2 2 3 2 4 4 5 4 6 6 7 RYGYRGY 2 2 1 7 G 2 4 1 1 Y 2 2 1 5 Y 2 4	2 2 3 -1
3 3 2 1 2 2 3 YRG 2 1 1 1 Y 4 3 1 2 2 3 2 4 RYGY 2 2 1 1 Y 2 4 5 8 1 2 2 3 3 4 3 5 YRYGY 2 3 1 5 G 2 1 1 2 Y 2 3 1 1 R 2 3 1 4 Y	-1 2 -1 2 -1 -1 3

Problem H. Sweet Remainders!

Time limit: 2 seconds
Memory limit: 1024 mebibytes

Given are integers $n, m > 0$, as well as integers $a_1, a_2, \ldots, a_n \in \{-1, 0, 1, \ldots, m-1\}$. Construct a tree T with n vertices $V = \{v_1, v_2, \ldots, v_n\}$ with the following condition: for each vertex v_i where $a_i \neq -1$, the remainder of the degree $\deg_T(v_i)$ when divided by m must equal a_i. If $a_i = -1$, then there are no restrictions on the degree of v_i.

Input

The first line of input contains an integer t, the number of test cases ($1 \leq t \leq 5 \cdot 10^5$). For each test case:

The first line contains two integers, n and m, denoting the number of vertices in the tree and the modulus ($1 \leq n \leq 5 \cdot 10^5$; $1 \leq m \leq 2 \cdot 10^9$).

The next line contains n integers $a_1, a_2, \ldots, a_n$ denoting the required remainders ($-1 \leq a_i < m$).

The sum of n across all test cases does not exceed $5 \cdot 10^5$.

Output

For each test case, if such a tree with n vertices exists, output "YES" (in any case), followed by $n-1$ lines with two integers each denoting the endpoints of each edge. Otherwise, output "NO" (in any case).

Example

standard input	*standard output*
6 1 5 0 2 3 1 1 5 4 0 1 1 1 1 6 2 1 0 0 0 0 1 3 2 1 1 1 4 10 3 3 1 1	YES YES 1 2 YES 1 2 1 3 1 4 1 5 YES 1 2 2 3 3 4 4 5 5 6 NO NO

Note

In the first test case, we are asked to build a tree with $n = 1$ vertex, $m = 5$, and degree residue array $b = [0]$. With only one vertex, there can be no edges, so there is exactly one possible tree: a single vertex of degree 0. Since $0 \bmod 5 = 0$, it satisfies the requirement; hence the answer is "YES".

In the second test case, we need a tree on $n = 2$ vertices with $m = 3$ and residues $b = [1, 1]$, meaning both degrees must be congruent to 1 modulo 3. On two vertices, there is only one tree: the single edge $1-2$. Its degrees are $(1, 1)$ and indeed $1 \bmod 3 = 1$, so the answer is "YES", and printing "1 2" is valid.

In the third test case, we need $n = 5$, $m = 4$, with residues $b = [0, 1, 1, 1, 1]$: the first vertex must have degree divisible by 4, while the other four vertices must have degree $\equiv 1 \pmod 4$. A natural construction is a star centered at vertex 1: connect 1 to every other vertex. Then $\deg(1) = 4$ and $4 \bmod 4 = 0$, while

each of $2, 3, 4, 5$ has degree 1 and $1 \bmod 4 = 1$. Therefore, the answer is "`YES`", and edges "`1 2`", "`1 3`", "`1 4`", "`1 5`" work.

In the fourth test case, we need a tree with $n = 6$, $m = 2$, residues $b = [1, 0, 0, 0, 0, 1]$, i.e. the endpoints must have odd degree and the four middle vertices must have even degree. Modulo 2, this is purely a parity constraint. The simple path $1-2-3-4-5-6$ fits perfectly: vertices 1 and 6 have degree 1 (odd), and vertices $2, 3, 4, 5$ have degree 2 (even). Hence the answer is "`YES`", and the path edges are correct.

In the fifth test case, we are asked to build a graph (and thus, in particular, a tree) on $n = 3$ with $m = 2$ and $b = [1, 1, 1]$, so every vertex must have odd degree. This is impossible: in every undirected graph, the number of odd-degree vertices is always even, so having three odd degrees cannot happen for any set of edges. Therefore, the verdict is "`NO`".

In the sixth test case, we have $n = 4$, $m = 10$, $b = [3, 3, 1, 1]$. Since $m > n - 1 = 3$, the residue modulo 10 equals the degree itself, so we require exactly $\deg = [3, 3, 1, 1]$. That forces two vertices to be adjacent to all others (degree 3 in a 4-vertex graph), and when $n > 2$, any such pair necessarily forms a cycle together with any third vertex. A tree cannot contain cycles, so the verdict is also "`NO`".

Problem I. Wooden Checker

Time limit: 2 seconds
Memory limit: 1024 mebibytes

This is an interactive problem.

Vlad has an empty directed graph with n vertices labeled by integers from 1 to n. He plans to add $n-1$ edges to this graph one by one, ensuring that at every moment two conditions are met:

1. The graph is a directed forest: each connected component is in the form of a directed rooted tree, where the edges are directed from the root to the leaves.
2. For each vertex of the graph, the set of vertices reachable from this vertex is a segment of integers without gaps.

Your task is to check these two conditions after adding each edge.

Interaction Protocol

First, the jury program will provide a string with an integer n ($1 \le n \le 2 \cdot 10^5$), the number of vertices.

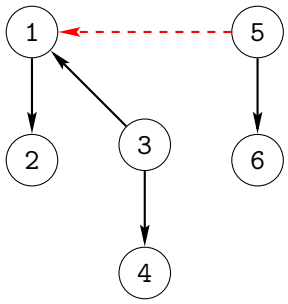
Then, there will be $n-1$ edge additions. For each addition, the jury will input a string with the vertex numbers v and u ($1 \le v, u \le n$; $v \ne u$). Check whether Vlad's conditions are met after adding the edge $v \to u$, and output the answer on a separate line in the following format.

- If the first condition is not met, output the string "Bad oriented forest".
- If the second condition is not met, output the string "Bad segment at v", where v is the number of any vertex for which the condition is not met.
- If both conditions are not met, output any of the two messages described above.
- Otherwise, if both conditions are met, output the string "Good".

If any condition is not met, your program should terminate immediately after outputting the message about it, even if it processed less than $n-1$ edges.

No directed edge $v \to u$ is added twice. All $n-1$ edge additions are fixed in advance.

Example

standard input	*standard output*	Notes
6 1 2 3 4 3 1 5 6 5 1	Good Good Good Good Bad segment at 5	

This page is intentionally left blank

Problem J. Euler Line

Time limit:	1 second
Memory limit:	1024 mebibytes

Given a line on the plane, construct a triangle with vertices at integer points such that its Euler line coincides with the given line.

Recall the basic definitions. Let $\triangle ABC$ be a non-degenerate triangle on the plane, M_A, M_B, and M_C be the midpoints of sides BC, AC, and AB, respectively, and s_A, s_B, and s_C be the *perpendicular bisectors* to sides BC, AC, and AB. Formally, s_A is the perpendicular to line BC, drawn at point M_A; lines s_B and s_C are defined similarly. From school geometry, it is known that lines s_A, s_B, and s_C intersect at a single point O. The point O has a remarkable property, namely, it is equidistant from all three vertices A, B, and C. The circle centered at O with radius OA is called the *circumcircle* of triangle ABC, and it is the unique circle in the plane that passes through all three vertices of the triangle. The point O is referred to as the *circumcenter* of triangle ABC.

Let H_A, H_B, and H_C denote the orthogonal projections of vertices A, B, and C onto lines BC, AC, and AB, respectively. The segments AH_A, BH_B, and CH_C are called the *heights* of triangle ABC. From school geometry, it is known that lines AH_A, BH_B, and CH_C intersect at a single point H, called the *orthocenter* of triangle ABC. In an acute or right triangle, H lies on the heights themselves, while in an obtuse triangle, it lies on their extensions.

The segments AM_A, BM_B, and CM_C are called the *medians* of the triangle. From school geometry, it is known that the medians intersect at a single point G, called the *centroid* or *barycenter* of triangle ABC.

The triangle $M_AM_BM_C$ is called the *medial* triangle for triangle ABC. The concepts above also apply to it: thus, point G is the barycenter of the medial triangle, and O is the orthocenter of the medial triangle. The circumcircle of triangle $M_AM_BM_C$ passes through M_A, M_B, M_C, H_A, H_B, H_C, as well as through the midpoints of segments AH, BH, CH, which is why it is called the *nine-point circle* of triangle ABC, and its center is denoted as O_9.

In an equilateral triangle, points O, H, G, and O_9 coincide. In any other triangle, they are pairwise distinct, but they lie on a single line, called the *Euler line* of triangle ABC. The arrangement of these four points on the Euler line is also known: O_9 is always the midpoint of segment OH, and point G lies on segment OH and divides it in the ratio $2:1$ (with G being twice as close to O as to H).

You are required to construct a non-degenerate triangle ABC with vertices at integer points, such that O, H, M, and O_9 lie on the given line, or report that this is impossible.

Input

The first line contains an integer t, the number of test cases ($1 \le t \le 5 \cdot 10^5$).

In the only line of the test case description, there are three integers, k, ℓ, and m, denoting the coefficients of the line ($-10^3 \le k, \ell, m \le 10^3$). The line will be defined as the set of points (x, y) that satisfy the equation $kx + \ell y + m = 0$. It is guaranteed that $k^2 + \ell^2 > 0$.

Output

For each test case, output six integers A_x, A_y, B_x, B_y, C_x, C_y, denoting the coordinates of the vertices of a non-degenerate triangle. The conditions $-10^6 \le A_x, A_y, B_x, B_y, C_x, C_y \le 10^6$ must be satisfied. The Euler line of triangle ABC must have the equation $kx + \ell y + m = 0$. If such a triangle does not exist, output six zeros separated by spaces.

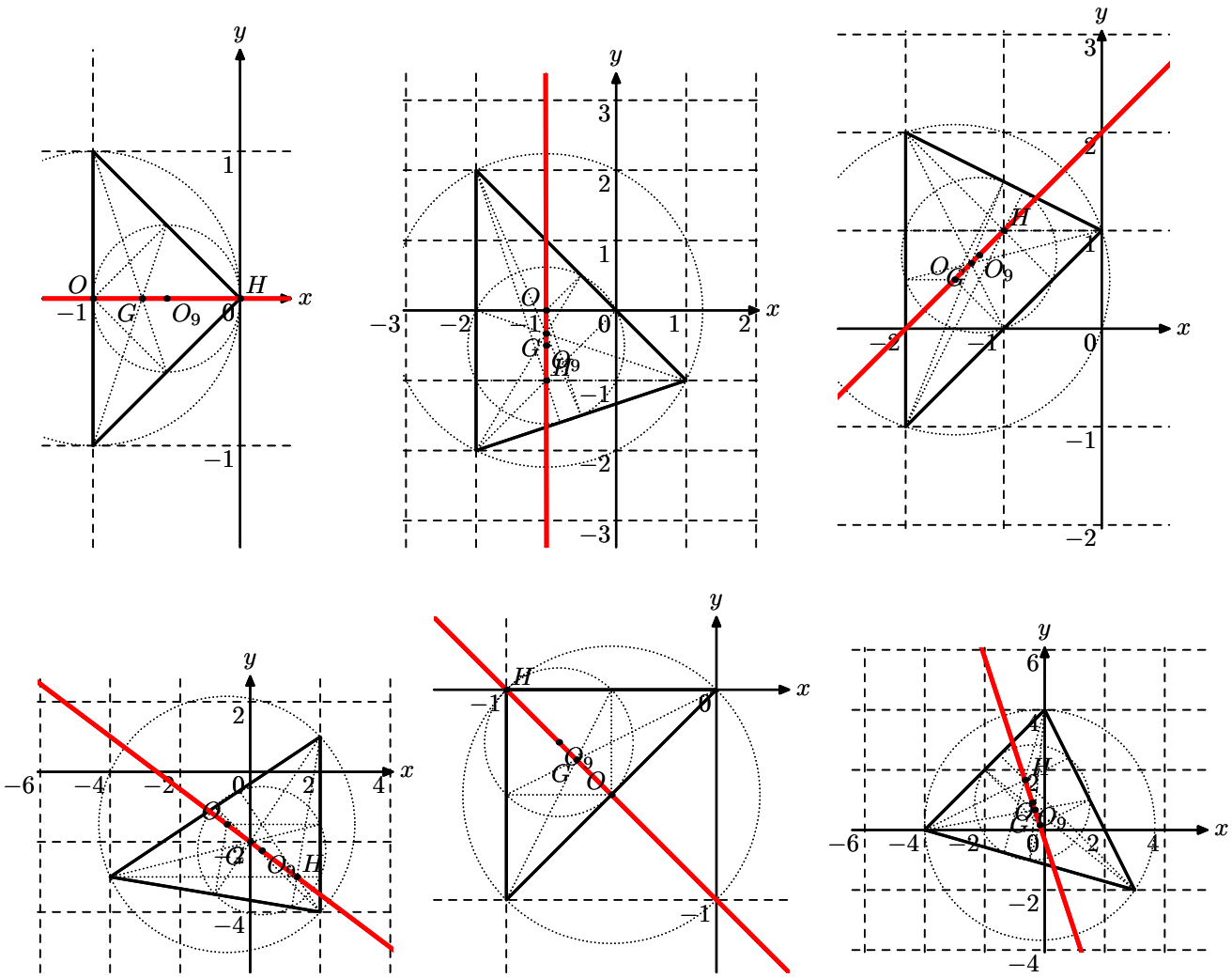
In each of the tests provided by the jury, it is guaranteed that, if there exists at least one triangle ABC with integral vertices with the given Euler line, there is also a triangle with conditions $-10^6 \le A_x, A_y, B_x, B_y, C_x, C_y \le 10^6$ satisfied.

Example

standard input	*standard output*
8 0 1 0 1 0 1 -1 1 -2 3 4 8 1 1 1 8 4 2 27 9 3 -544 862 12	-1 -1 -1 1 0 0 -2 -2 -2 2 1 -1 -2 -1 -2 2 0 1 -4 -3 2 -4 2 1 -1 -1 -1 0 0 0 0 0 0 0 0 0 -4 0 3 -2 0 4 5811 -1154 -3261 -2058 -3713 2478

Note

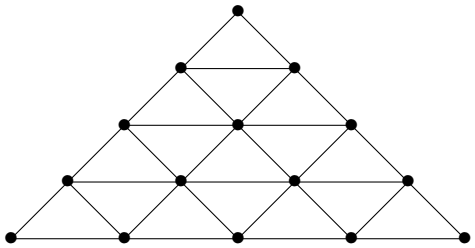
We do not explicitly require $\triangle ABC$ to be non-equilateral because it is impossible to come up with an equilateral triangle whose vertices have integer coordinates.



Problem K. Traversal of a Triangular Grid

Time limit: 2 seconds
Memory limit: 1024 mebibytes

Let $n \geqslant 1$. Consider the following standard way to divide a large triangle into n^2 smaller triangles by drawing three sets of pairwise parallel lines:



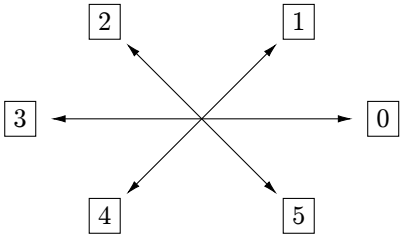
Naturally, this defines an undirected graph with $(n+1)(n+2)/2$ vertices and $3n(n+1)/2$ edges: the vertices are the nodes of the grid or, in other words, the intersection points of the lines (marked in bold in the picture), and the edges are the segments between neighboring nodes. Your task is to construct an Eulerian cycle on the edges of this graph that "starts" from the topmost vertex of the triangle. In other words, you need to start at the topmost vertex, move along the edges, traverse each edge exactly once, and, in the end, return to the topmost vertex. It can be proven that a solution always exists.

Input

An integer n is given, ranging from 1 to 20.

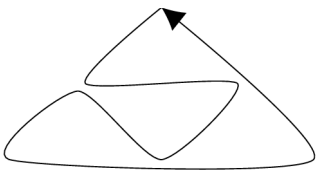
Output

Output a string of exactly $3n(n+1)/2$ digits from 0 to 5, representing some Eulerian cycle of the graph in the following format: 0 means moving directly to the right from the current vertex, 1 means moving in the right-up direction, 2 means moving left-up, 3 means moving left, 4 means moving left-down, and 5 means moving right-down. For clarity, the directions are shown in the picture below.



The traversal **must start from the top vertex**. Any of the possible traversals will be accepted.

Example

standard input	*standard output*	*picture of the path*
2	404240022	

This page is intentionally left blank

Problem L. Triangles

Time limit: 2 seconds
Memory limit: 1024 mebibytes

Consider n integer points on a plane, where n is divisible by 3. The x coordinates are a permutation of integers from 1 to n. The y coordinates are also a permutation of integers from 1 to n.

Construct a configuration of $n/3$ triangles with vertices in the given points: each point must appear as a vertex of one of the triangles. The triangles can be degenerate and can intersect freely.

The score of a configuration is the sum of the sizes of all triangles in it. The size of a triangle is the perimeter of its bounding rectangle. The bounding rectangle is the minimal rectangle with sides parallel to coordinate axes that contains the triangle.

Find a configuration that achieves the maximum possible score.

Input

The first line contains an integer n ($3 \le n \le 99\,999$; n is divisible by 3).

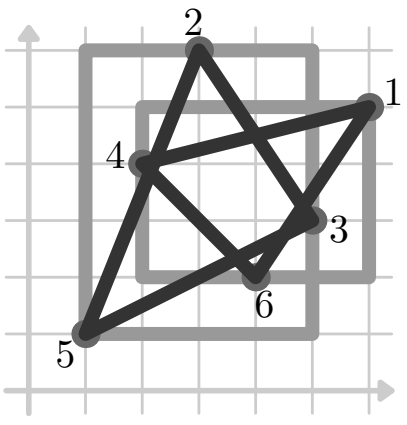
Each of the next n lines contains two integers x and y: the coordinates of one of the points. Each integer from 1 to n appears once among the x coordinates and once among the y coordinates.

Output

On the first line, print the maximum possible score.

Next, print any configuration that achieves this score. For that, print $n/3$ lines with three integers in each: the numbers of points in the input that are the vertices of a triangle. The points are numbered from 1 to n in the input order. Triangles and their vertices can be printed in any order.

Example

standard input	*standard output*	*explanation*
6 6 5 3 6 5 3 2 4 1 1 4 2	32 2 3 5 6 1 4	

This page is intentionally left blank

Problem M. Construction Company

Time limit:	12 seconds
Memory limit:	1024 mebibytes

The New Year holidays are over, and your construction company is getting back to work. Unfortunately, some employees are still celebrating...

Today, a sober and b drunk workers came to work for you. Each worker will be available for the whole day. Your company also received m orders for today. Each order is specified by the start and the end of the time span to perform it: $[st, fn]$. Such an order requires a single worker to work on it from moment st to moment fn, inclusive. A worker can complete multiple orders if their time spans do not intersect. Additionally, each order has one of the two types: a simple order can be completed by any worker, while a complex order requires a sober worker.

Will your company be able to handle today's orders?

Input

The first line contains an integer t, the number of test cases ($1 \le t \le 2000$). For each test case:

The first line contains three integers: a, b, and m ($0 \le a, b, m \le 10\,000$).

Then m lines follow. Each line contains three integers, *type*, st, and fn, describing a simple (if *type* $= 1$) or complex (*type* $= 2$) order with the time span $[st, fn]$ (*type* $\in \{1, 2\}$; $1 \le st \le fn \le 2 \cdot m$). For example, a line `1 2 4` describes a simple order which requires three moments, from second to fourth, both ends included.

The sum of m over all test cases is at most 10 000.

Output

For each test case, print a single line with the answer: "`Yes`" or "`No`" (case-insensitive).

Example

standard input	*standard output*
1 1 1 2 1 1 2 2 2 4	Yes

This page is intentionally left blank