Problem A. Adjacent Seats

Input file:	`stdin`
Output file:	`stdout`
Time limit:	2 seconds
Memory limit:	512 megabytes

Let $N \geq 3$ be an odd number.

There are N seats arranged in a circle. The seats are numbered 0 through $N-1$. For each i $(0 \leq i \leq N-2)$, seat i and seat $i+1$ are adjacent. Also, seat $N-1$ and seat 0 are adjacent.

Each seat is either vacant, or occupied by a man or a woman. However, no two adjacent seats are occupied by two people of the same sex. It can be shown that there is at least one empty seat because N is an odd number greater than 1.

You are given N, but the states of the seats are not given. Your objective is to correctly guess the ID number of any one of the empty seats. To do so, you can repeatedly send the following query:

Choose an integer i $(0 \leq i \leq N-1)$. If Seat i is empty, the problem is solved. Otherwise, you are notified of the sex of the person at seat i.

Guess the ID number of an empty seat by sending at most 20 queries.

Interaction Protocol

In the first line you are given one integer N – the number of seats (N is odd, $3 \leq N \leq 99\,999$).

After that, you start to send queries. A query consists of one integer i $(0 \leq i \leq N-1)$ – number of seat.

Do not forget to end the query by end-of-line character and to flush standard output after each query.

The response of the interactor is one of three possible answers: "`Vacant`", "`Male`" and "`Female`". Each of these means that Seat i is empty, occupied by a man and occupied by a woman, respectively.

When you receive "`Vacant`" answer, immediately terminate the program. If you send more than 20 queries, you will receive the Wrong Answer verdict.

Example

stdin	stdout
3 Male Female Vacant	0 1 2

Explanation

In the sample, $N = 3$, and Seat $0, 1, 2$ are occupied by a man, occupied by a woman and vacant, respectively.

Problem B. Broken Sequence

Input file:	`stdin`
Output file:	`stdout`
Time limit:	2 seconds
Memory limit:	512 megabytes

You have a sequence of integers and there used to be operators between them. However, after an unexpected Windows Update, the operators all magically disappeared!

$$A_1?A_2?A_3?\ldots?A_N$$

Due to the lack of stability of Windows 18H2, the content of the sequence may change overtime, and these changes are permanent.

You know these operators may be $+$, $-$ or $\times$. Every possible combination, which is 3^{n-1} in total, can be evaluated and it's your job to calculate the sum of the evaluation results.

Input

The first line of input contains integers N, Q ($1 \le N, Q \le 100\ 000$), the number of integers and queries.

The second line contains N integers and the i_{th} integer represents A_i.

Each of the following Q lines contains 2 integers T_i ($1 \le T_i \le N$) and V_i indicating a query changing A_{T_i} to V_i.

It is guaranteed that $0 \le A_i, V_i \le 10\ 000$.

The problem only has 3 tests, other than the samples.

Output

You shuold output Q lines. The i_{th} line contains the number representing the answer after the i_{th} query module 1 000 000 007.

Example

stdin	stdout

2 1
1 1
2 2
	4

5 5
9384 887 2778 6916 7794
2 8336
5 493
3 1422
1 28
4 60
	890543652
	252923708
	942282590
	228728040
	608998099

Notes

Possible combination of the first example.

$$1+2=3 \quad 1\times 2=2 \quad 1-2=-1$$

Problem C. Casino Cheating

Input file:	`stdin`
Output file:	`stdout`
Time limit:	1 second
Memory limit:	512 megabytes

Casino "Three Devices for Lumbering" has come up with a new gambling game which has immediately received much attention.

The rules of this game are very easy. Two players alternate their turns in the game: a visitor and a croupier. At the start of the round, the croupier has a whole chocolate bar, and the visitor goes first. In his turn, each player takes any of the pieces of chocolate which belong to the opponent, cuts it into two pieces so that the bigger piece is at most twice as large as the smaller. After that, he takes one of these two pieces for himself and leaves the other one. The casino decides not to investigate any optimal strategies in this game. Instead, the croupier in his turn chooses one of the visitor's pieces with equal probability, and takes as much as possible: $\frac{2}{3}$ of this piece.

The randomness of the croupier's turns should convince the players that the croupier will make random and wrong turns. The game lasts for a fixed odd number of turns: as the visitor starts without any chocolate at all, and croupier has a whole chocolate bar, it is guaranteed by the rules that the visitor makes the first and the last turns.

The cost of participation in one round is rather high: the player must pay 0.55 chocolate units, or c.u., which will not be refunded after the game. Despite this fact, analysts of a rival company have found a vulnerability in the rules which can guarantee making profit from each round of the game: get at least 0.55 of the chocolate piece in total in each round.

You have to develop such a strategy which gains at least 0.55 of the chocolate piece in total in each round, before the casino closes this vulnerability.

Interaction Protocol

In the first line, you are given two integers t and n: the number of rounds in which you must win and the number of turns that will be made in each round ($1 \le t \le 1000$; $1 \le n \le 30$; n is odd).

After that, you must win t rounds of the game. At the start of each round, the croupier has a whole chocolate bar which is labeled by 0. Each next piece of chocolate is labeled with the number of the turn when a player cuts this piece from the opponent's piece and takes it for himself. There will be n turns, the first and the last will be made by your program.

In your turn, you choose a piece of chocolate with some label i (i should be even because all the croupier's pieces have even numbers, and do not exceed the number of the current turn), which belongs to the croupier, and the percentage x ($\frac{1}{3} \le x \le \frac{2}{3}$) of this piece which you take for yourself. When you choose i and x, you should print these two numbers. The number x should be printed with at least ten digits after the decimal point (the more the better). If your x is less than $\frac{1}{3}$, it will be considered as $\frac{1}{3}$, and if it is bigger than $\frac{2}{3}$, it will be considered as $\frac{2}{3}$. In his turn, the croupier randomly chooses one of your pieces, a uniformly distributed random odd i not exceeding the number of the current turn, takes $\frac{2}{3}$ of this piece for himself, and prints these numbers. The fraction $\frac{2}{3}$ is always printed as `0.6666666667`.

After your last turn, the interactor checks if you have at least $0.55 - 10^{-9}$ of the initial chocolate. If not, the interactor prints 0, and your program should terminate: you don't pass the test and get 'Wrong Answer' on this test. Otherwise, the interactor prints 1, and a new round starts immediately, where the interactor is waiting for your first turn again.

Example

stdin	stdout
2 5	0 0.5
1 0.6666666667	2 0.6666666667
3 0.6666666667	0 0.6666666667
1	0 0.6666666667
1 0.6666666667	2 0.6666666667
3 0.6666666667	0 0.6666666667
0	

Notes

Let's take a look at what happens in the first sample. The participant must win in two rounds.

The first round:

- Before the first turn, the participant doesn't have any pieces of chocolate, and the jury has one piece of size 1.
- After the first turn, the participant has one piece of size $\frac{1}{2}$, and the jury has one piece of size $\frac{1}{2}$.
- After the second turn, the participant has one piece of size $\frac{1}{6}$, and the jury has pieces of sizes $\frac{1}{2}$ and $\frac{1}{3}$.
- After the third turn, the participant has pieces of sizes $\frac{1}{6}$ and $\frac{2}{9}$, and the jury has pieces of sizes $\frac{1}{2}$ and $\frac{1}{9}$.
- After the fourth turn, the participant has pieces of sizes $\frac{1}{6}$ and $\frac{2}{27}$, and the jury has pieces of sizes $\frac{1}{2}$, $\frac{1}{9}$ and $\frac{4}{27}$.
- After the fifth turn, the participant has pieces of sizes $\frac{1}{6}$, $\frac{2}{27}$ and $\frac{1}{3}$, and the jury has pieces of sizes $\frac{1}{6}$, $\frac{1}{9}$ and $\frac{4}{27}$.

At the end, the participant has $\frac{1}{6} + \frac{2}{27} + \frac{1}{3} \approx 0.574$ which is bigger than 0.55: the participant wins the round.

The second round:

- Before the first turn, the participant doesn't have any pieces of chocolate, and the jury has one piece of size 1.
- After the first turn, the participant has one piece of size $\frac{2}{3}$, and the jury has one piece of size $\frac{1}{3}$.
- After the second turn, the participant has one piece of size $\frac{2}{9}$, and the jury has pieces of sizes $\frac{1}{3}$ and $\frac{4}{9}$.
- After the third turn, the participant has pieces of sizes $\frac{2}{9}$ and $\frac{8}{27}$, and the jury has pieces of sizes $\frac{1}{3}$ and $\frac{2}{27}$.
- After the fourth turn, the participant has pieces of sizes $\frac{2}{9}$ and $\frac{8}{81}$, and the jury has pieces of sizes $\frac{1}{3}$, $\frac{2}{27}$ and $\frac{16}{81}$.
- After the fifth turn, the participant has pieces of sizes $\frac{2}{9}$, $\frac{8}{81}$ and $\frac{2}{9}$, and the jury has pieces of sizes $\frac{1}{9}$, $\frac{2}{27}$ and $\frac{16}{81}$.

At the end, the participant has $\frac{2}{9} + \frac{8}{81} + \frac{2}{9} \approx 0.543$ which is less than 0.55: the participant loses the round, and the interactor prints 0, which means that solution got "Wrong Answer" on this test.

Problem D. Deprecated Template Book

Input file:	`stdin`
Output file:	`stdout`
Time limit:	3 seconds
Memory limit:	512 megabytes

It's well-known that Oxx has a very powerful template book that he wants to bring to the World Finals. However, the template book might be deprecated in the year 8102. To test the authenticity of the book, he proposed the following problem and claimed that you could never get Accepted in case your book were a fake one.

The **Oxx Spacecraft**, which was built with 10 billion yuan and launched in October 8102, in Qingdao, is staring continuously at a patch of sky containing about 17 rare stars in the **WF19** constellation. We are careful to set up the universe as a three dimensional space (x, y, z). The exact location of our planet is (a, b, c) and the center of **WF19** constellation is (u, v, w). The ground detector recorded the initial state of Quapler. It faces the x direction and the y direction is on the left side.

The flight recorder captured its all orbit transformations with a total of n in a timed sequence. Each record as a triple (d, s, t) describes a rectilinear motion along the current direction of length d and ending with a directional rotation.

The string s is one of "R", "L", "D", "U" indicating a rotation with t radian to the right, the left, the down and the up side respectively. The recorded information is based on relative direction. That is say, the exact meaning of "R", "L", "D" and "U" should be discussed in the current direction of **Oxx Spacecraft** spacecraft itself.

To emphasize the importance of the studying, the scientists need to analyze the shortest distance between **Oxx Spacecraft** and the target constellation **WF19**.

Input

The first line consisting a number T $(1 \le T \le 30\,000)$ indicating the number of test cases.

For each test case, the first line contains three float numbers a, b and c where $0 \le a, b, c \le 100$. The second line contains three float numbers u, v and w where $0 \le u, v, w \le 100$. The third line is the total number of orbit transformation n $(1 \le n \le 30)$. Each of the following n lines consists a float number d $(0 \le d \le 100)$, a string s and a float number t $(0 \le t \le \pi + 10^{-4})$ described as above.

All float numbers given in the input is has at most 4 digits after the decimal point.

Output

For each test case, print the shortest distance.

The answer is correct with absolute error less than 10^{-2}.

Note that the Spacecraft might go through the constellation, in which case the answer should be 0.

Example

stdin	stdout
1	1.41
0 0 0	
1 1 1	
7	
2 U 1.5708	
2 U 1.5708	
2 L 1.5708	
2 L 1.5708	
2 U 1.5708	
2 U 1.5708	
2 L 1.5708	
3	63.7573
98.0663 29.2686 4.0475	28.2537
57.1690 9.4575 48.7680	83.3797
2	
42.6416 D 0.6391	
92.5603 D 0.9831	
24.2912 41.9040 34.3073	
54.1783 11.1332 12.7184	
4	
27.4207 R 2.1224	
61.1714 R 1.2875	
67.7295 D 3.0435	
61.2343 U 1.2411	
79.0860 63.4667 8.9258	
24.7665 9.3323 61.5733	
3	
60.9123 L 3.0820	
90.0900 L 2.8897	
73.7178 R 1.5021	

Problem E. Excellent Programmer

Input file:	`stdin`
Output file:	`stdout`
Time limit:	10 seconds
Memory limit:	512 megabytes

A countless number of skills are required to be an excellent programmer. Different skills have different importance degrees, and the total programming competence is measured by the sum of products of levels and importance degrees of his/her skills.

In this summer season, you are planning to attend a summer programming school. The school offers courses for many of such skills. Attending a course for a skill, your level of the skill will be improved in proportion to the tuition paid, one level per one yen of tuition, however, each skill has its upper limit of the level and spending more money will never improve the skill level further. Skills are not independent: For taking a course for a skill, except for the most basic course, you have to have at least a certain level of its prerequisite skill.

You want to realize the highest possible programming competence measure within your limited budget for tuition fees.

Input

The first line of the input contains T ($1 \le T \le 100$). Then follows T test cases. For each test case:

The first line has two integers, n, the number of different skills between 2 and 100, inclusive, and k, the budget amount available between 1 and 10^5, inclusive. In what follows, skills are numbered 1 through n.

The second line has n integers $h_1, \ldots, h_n$, in which h_i is the maximum level of the skill i, between 1 and 10^5, inclusive.

The third line has n integers $s_1, \ldots, s_n$, in which s_i is the importance degree of the skill i, between 1 and 10^9, inclusive.

The fourth line has $n-1$ integers $p_2, \ldots, p_n$, in which p_i is the prerequisite skill of the skill i, between 1 and $i-1$, inclusive. The skill 1 has no prerequisites.

The fifth line has $n-1$ integers $l_2, \ldots, l_n$, in which l_i is the least level of prerequisite skill p_i required to learn the skill i, between 1 and h_{p_i} , inclusive.

Output

For each dataset, output a single line containing one integer, which is the highest programming competence measure achievable, that is, the maximum sum of the products of levels and importance degrees of the skills, within the given tuition budget, starting with level zero for all the skills. You do not have to use up all the budget.

Example

stdin	stdout
3	18
3 10	108
5 4 3	35
1 2 3	
1 2	
5 2	
5 40	
10 10 10 10 8	
1 2 3 4 5	
1 1 2 3	
10 10 10 10	
5 10	
2 2 2 5 2	
2 2 2 5 2	
1 1 2 2	
2 1 2 1	

Problem F. Freshman's Dream

Input file:	`stdin`
Output file:	`stdout`
Time limit:	6 seconds
Memory limit:	1024 megabytes

Freshman is a junior CS student. He doesn't know anything about NP-Hard problems. So even if there is an NP-Hard problems in the assignment, which can be possibly left by the professor on purpose to test the students, he still thinks about it (during sleep of course), and, sometimes comes up with some brilliant ideas.

There is a bidirectional graph consisting of n vertices and m edges. The vertices and edges are numbered from 1 to n and 1 to m respectively, and the weight of edge i is w_i $(1 \leq i \leq m)$. Given a natural number k, find the length of the shortest simple path that starts from vertex 1 and ends at vertex n, and consists of k edges. A simple path is a path that does not visit same vertex twice, and length of a path is the sum of weight of edges that consists the path.

Input

In the first line, three space-separated integers n, m, k are given. $(2 \leq n \leq 10^6$, $1 \leq m, k \leq 10^6$, $\min(n, m, k) \leq 5)$.

In the next m lines, three space-separated integers x_i, y_i, w_i are given. They denote that edge i is connecting vertex x_i and vertex y_i, and has weight w_i. $(1 \leq x_i, y_i \leq n$, $1 \leq w_i \leq 10^8)$

No loops or multiple edges are given.

Output

Print the length of the shortest simple path that starts from vertex 1 and ends at vertex n, and consists of k edges. If there is no such path, print -1.

Example

stdin	stdout

6 6 3
1 2 3
2 3 1
3 6 4
1 4 1
4 5 5
5 6 9
	8

Problem G. Generate Problems

Input file:	`stdin`
Output file:	`stdout`
Time limit:	3 seconds
Memory limit:	512 megabytes

Well-tagged Problems

When ultmaster generates a problem, she will look up the Tree of Tags to make sure the problem is well tagged. The tag tree is a rooted tree, which describes the inheritance relationship of the tags. For example, Shortest Path and MST are children nodes of Graph Theory.

Now ultmaster needs to generate M problems for some secret contest and every problem needs to have at least L tags and at most R tags, so that the problem is neither too easy nor too hard. The tags should be **consecutive** nodes on the tag tree and must be ancestors of each other. For example, a problem tagged DP and Graph is not allowed. In other words, **the tags of a problem should be a chain** with their lowest common ancestor **(LCA) on one end**.

In a problem set, problems with identical sets of tags are prohibited.

Tags have interesting values and the interesting value of a problem is the sum of its tags' value. For example, Dynamic programming may have a value of 10, and plug DP may have one of -5.

Ultmaster wants to maximize the sum of values of the problems she generates. Can you help her?

Input

The first line contains a single integer N ($2 \le N \le 500\,000$) indicating number of tags.

The second line contains N integers and the i_{th} integer f_i indicates the parent tag of i_{th} tag is f_i. Particularly, $f_i = 0$ means the i_{th} tag is the root tag (Root of all evil). We guarantee the inputs describes a single rooted tree. For simplicity, the tests ensure that $f_1 = 0$ and $f_i < i$ for $i \in [2, N]$.

The third line contains N integers and the i_{th} integer a_i ($|a_i| \le 2000$) represents the value of the i_{th} tag.

The forth line contains 3 integers M ($1 \le M \le 500\,000$), L, R ($1 \le L \le R \le N$) represent the number of required problems and the upper and lower bound of tags used on the single problem.

Output

Output a single number indicating the maximal sum.

Example

stdin	stdout

7
0 1 1 2 2 3 3
2 3 4 1 2 3 4
3 3 3
	26

Notes

In the sample test case, ultmaster will generate 3 problems: one with tag 1, 3, 7, one with tag 1, 3, 6 and one with tag 1, 2, 5.

Problem H. Head-tail Palindromes

Input file:	`stdin`
Output file:	`stdout`
Time limit:	2 seconds
Memory limit:	512 megabytes

A palindrome is a string that reads the same from left to right as it does from right to left.

Given a string, S, of N lowercase English letters, we define a k-length rotation as cutting the first k characters from the beginning of S and appending them to the end of S. For each S, there are N possible k-length rotations (where $0 \leq k < N$).

Given N and S, find all k-length rotations of S; for each rotated string, S_k, print the maximum possible length of any palindromic substring of S_k on a new line.

Input

The first line contains an integer, N (the length of S). The second line contains a single string, S.

Output

There should be N lines of output, where each line k contains an integer denoting the maximum length of any palindromic substring of rotation S_k.

Constraints

- $1 \leq N \leq 5 \times 10^5$
- $0 \leq k < N$
- S is comprised of lowercase English letters.

Explanation

The examples are on the next page.

Consider Sample Case 2, where $S = cacbbba$.

The possible rotations, S_k, for string S are:

- $S_0 = cacbbba$.
- $S_1 = acbbbac$.
- $S_2 = cbbbaca$.
- $S_3 = bbbacac$.
- $S_4 = bbacacb$.
- $S_5 = bacacbb$.
- $S_6 = acacbbb$.

The longest palindromic substrings for each S_k are:

- S_0: "*cac*" and "*bbb*", so we print their length (3) on a new line.
- S_1: "*bbb*", so we print its length (3) on a new line.

- S_2: "*bbb*" and "*aca*", so we print their length (3) on a new line.
- S_3: "*bbb*", "*aca*" and "*cac*", so we print their length (3) on a new line.
- S_4: "*aca*" and "*cac*", so we print their length (3) on a new line.
- S_5: "*aca*" and "*cac*", so we print their length (3) on a new line.
- S_6: "*aca*", "*cac*" and "*bbb*", so we print their length (3) on a new line.

Example

stdin	stdout
13 aaaaabbbbaaaa	12 12 10 8 8 9 11 13 11 9 8 8 10
7 cacbbba	3 3 3 3 3 3 3
12 eededdeedede	5 7 7 7 7 9 9 9 9 7 5 4

Problem I. Interesting Nim Game

Input file:	`stdin`
Output file:	`stdout`
Time limit:	2 seconds
Memory limit:	512 megabytes

Alice and Bob are playing the game of Nim with n piles of stones with sizes $p_0, p_1, \ldots, p_{n-1}$. If Alice plays first, she loses if and only if the 'xor sum' (or 'Nim sum') of the piles is zero, i.e., $p_0 \oplus p_1 \oplus \cdots \oplus p_{n-1} = 0$.

Since Bob already knows who will win (assuming optimal play), he decides to cheat by removing some stones in some piles before the game starts. However, to reduce the risk of suspicion, he must keep at least one pile unchanged. Your task is to count the number of ways Bob can remove the stones to force Alice into losing the game. Since the number can be very large, output the number of ways modulo $10^9 + 7$. Assume that both players will try to optimize their strategy and try to win the game.

Input

The first line of the input contains an integer n denoting the number of piles. The next line contains n space-separated integers $p_0, p_1, \ldots, p_{n-1}$ indicating the sizes of the stone piles.

Constraints

- $3 \le n \le 100$
- $0 \le p_i \le 10^9$

Output

Print a single integer denoting the number of ways Bob can force Alice to lose the game, modulo $10^9 + 7$.

Example

stdin	stdout
3 1 2 3	4
10 10 10 1 1 1 1 1 10 10 10	321616

Explanation

For example 1:

- $[0, 2, 2]$
- $[1, 0, 1]$
- $[1, 1, 0]$
- $[1, 2, 3]$

Problem J. Jump

Input file:	`stdin`
Output file:	`stdout`
Time limit:	2 seconds
Memory limit:	512 megabytes

Consider a toy interactive problem OneMax which is defined as follows. You know an integer n and there is a hidden bit string S of length n. The only thing you may do is to present the system a bit string Q of length n, and the system will return the number $\text{OneMax}(Q)$ — the number of bits which coincide in Q and S at the corresponding positions. The name of OneMax problem stems from the fact that this problem is simpler to explain when $S = 111\ldots11$, so that the problem turns into maximization (Max) of the number of ones (One).

When n is even, there is a similar (but harder) interactive problem called Jump. The simplest way to describe the Jump is by using OneMax:

$$\text{Jump}(Q) = \begin{cases} \text{OneMax}(Q) & \text{if } \text{OneMax}(Q) = n \text{ or } \text{OneMax}(Q) = n/2; \\ 0 & \text{otherwise.} \end{cases}$$

Basically, the only nonzero values of OneMax which you can see with Jump are n (which means you've found the hidden string S) and $n/2$.

Given an even integer n — the problem size, you have to solve the Jump problem for the hidden string S by making interactive Jump queries. Your task is to eventually make a query Q such that $Q = S$.

Interaction Protocol

First, the testing system tells the length of the bit string n. Then, your solution asks the queries and the system answers them as given by the Jump definition. When a solution asks the query Q such that $Q = S$, the system answers n and terminates, so if your solution, after reading the answer n, tries reading or writing anything, it will fail.

The limit on the number of queries is $n + 500$. If your solution asks a $(n + 501)$-th query, then you will receive the "Incorrect" outcome. You will also receive this outcome if your solution terminates too early.

If your query contains wrong characters (neither `0`, nor `1`), or has a wrong length (not equal to n), the system will terminate the testing and you will receive the "Incorrect" outcome.

Finally, if everything is OK (e.g. your solution finds the hidden string) on every test, you will receive the "Accepted" outcome, in this case you will have solved the problem.

Input

The first line of the input stream contains an even number n ($2 \le n \le 1000$). The next lines of the input stream consist of the answers to the corresponding queries. Each answer is an integer — either 0, $n/2$, or n. Each answer is on its own line.

Output

To make a query, print a line which contains a string of length n which consists of characters `0` and `1` only. Don't forget to put a newline character and to flush the output stream after you print your query.

Example

stdin	stdout
2 1 0 1 2	01 11 10 00

Problem K. Kidnapping

Input file:	`stdin`
Output file:	`stdout`
Time limit:	2 seconds
Memory limit:	512 megabytes

As unlikely as it may seem, a crazy guy on the phone claims to have kidnapped your precious child. You don't really believe him, as all your children (possibly none) are playing in front of you right now, safe and sound. Anyway, you're fairly curious about the situation, so you ask the criminal what he wants for releasing his hostage.

As boring as it may seem, the kidnapper asks for money. Just money. You are about to hang up the phone in disappointment when something peculiar attracts your attention. Your interlocutor is not telling you the exact amount he wants. Instead, he proposes you a riddle. As ridiculous as it may seem, the riddle is:

"How many non-empty sets of positive integers exist such that their greatest common divisor is 1*, while their least common multiple is* m*?"*

Then, the abductor tells you that the answer to this riddle, taken modulo 998244353, is the exact amount of money he wants for returning your imaginary offspring.

You're now wondering about the rates at the kidnapping market, since you've been away from this kind of affairs for quite some time. Not that you're going to pay the snatcher a single penny, though.

Input

The only line of the input contains a single integer m ($1 \le m \le 10^{18}$).

Output

Output a single integer – the amount of money you've been asked for.

Example

stdin	stdout
6	7
100	322

Notes

In the first example test case, all suitable sets are $\{1, 6\}$, $\{2, 3\}$, $\{1, 2, 3\}$, $\{1, 2, 6\}$, $\{1, 3, 6\}$, $\{2, 3, 6\}$, and $\{1, 2, 3, 6\}$.