# Problem A. Gruz Cable

| | |
|---|---|
| Input file: | `cable.in` |
| Output file: | `cable.out` |
| Time limit: | 6 seconds |
| Memory limit: | 256 mebibytes |

Boy Gruz has a long cable with $N$ marks on it. The cable lies on the ground and forms a straight line segment. The marks are labeled by integers from 1 to $N$ so that the leftmost mark is labeled by 1, the next mark to the right is labeled by 2, and so on. The rightmost mark is labeled by $N$. At each mark, there is one smaller electric wire of some color hanging on the cable.

Gruz can connect a pair of different wires if they share the same color. Each wire can be connected with at most one other wire. He wants to make as many connected pairs as possible, but there is one restriction: Gruz cannot connect a pair of wires if, upon doing so, some two pairs of wires will intersect.

To define intersecting pairs in a formal way, we will denote electric wires by the labels on the marks at which they hang on the cable. Two pairs $(a_1, b_1)$ (where $a_1 < b_1$) and $(a_2, b_2)$ (where $a_2 < b_2$) intersect if and only if $a_1 < a_2 < b_1 < b_2$ or $a_2 < a_1 < b_2 < b_1$. Note that connections $a_1 < a_2 < b_2 < b_1$ and $a_2 < a_1 < b_1 < b_2$ in which one pair is completely "inside" the other are allowed.

Gruz doesn't want to think, so he asks you to find the maximal number of pairs of wires which can be connected and output such pairs.

## Input

On the first line of input, there is an integer $N$, the number of marks on the cable ($1 \le N \le 1500$). The next line contains a single string consisting of characters with ASCII codes in range from 33 to 126, inclusive. This string represents the colors of the wires hanging on the cable from left to right. Different colors are represented by different characters, same colors by same characters.

## Output

On the first line of output, print $K$: the maximal possible number of connected pairs. In the next $K$ lines, output pairs of integers, one per line: pairs of connected wires where each wire is denoted by the label on its mark. In each pair, the first label should be less than the second one. Pairs can be printed in any order. If there are multiple solutions, output any one of them.

## Examples

| cable.in | cable.out |
|---|---|
| 6<br>abadbd | 2<br>1 3<br>4 6 |
| 4<br>aBBa | 2<br>1 4<br>2 3 |

# Problem B. Genealogy

| | |
|---|---|
| Input file: | `genealogy.in` |
| Output file: | `genealogy.out` |
| Time limit: | 7 seconds |
| Memory limit: | 256 mebibytes |

During a dispute in the Parliament, Lords with similar views on the problem usually unite in groups. The final resolution of the Parliament is almost always the resolution of the most important group. That is why it is extremely important to be able to calculate importance of a group.

The most important thing for each Lord is the antiquity of his family, so the importance of each Lord is equal to his family's antiquity. The antiquity of a Lord's family is the amount of the Lord's ancestors: his father, his grandfather, his great-grandfather and so on up to the founder of the family. To calculate importance of a group, you need to count the number of Lords in the group along with all their ancestors. Note that, if a certain Lord is the ancestor of two or more Lords in the group, that Lord should be counted only once.

You will be given the family tree of the Lords (surprisingly, all the Lords descend from one Lord) and the list of groups. For each given group, you should find and print the importance of that group.

## Input

The first line of input contains one integer $n$, the number of Lords ($1 \leq n \leq 100\,000$). The Lords are conveniently numbered by integers from 1 to $n$. The next line contains $n$ integers $p_1$, $p_2$, $\ldots$, $p_n$ where $p_i$ is the parent of Lord number $i$; if this Lord is the founder of the family, $p_i$ is equal to $-1$. It is guaranteed that this data represents a valid family tree.

The third line contains one integer $g$, the number of groups ($1 \leq g \leq 3\,000\,000$). Next $q$ lines contains description of groups. $j$-th of these lines contains an integer $k_j$, the size of $j$-th group ($1 \leq k_j \leq n$), followed by $k_j$ distinct integers which are the numbers of the Lords in group $j$. It is guaranteed that the sum of all $k_j$ does not exceed $3\,000\,000$.

## Output

Output $g$ lines. On the $j$-th line, print one integer: the importance of group $j$. It is guaranteed that the size of the output file will not exceed six mebibytes.

## Examples

| genealogy.in | genealogy.out |
|---|---|
| 4 | 4 |
| -1 1 2 3 | 4 |
| 4 | 4 |
| 1 4 | 4 |
| 2 3 4 | |
| 3 2 3 4 | |
| 4 1 2 3 4 | |
| 5 | 4 |
| 2 -1 1 2 3 | 4 |
| 10 | 5 |
| 3 3 4 1 | 2 |
| 3 2 4 3 | 3 |
| 4 1 3 5 4 | 4 |
| 1 4 | 1 |
| 2 2 3 | 5 |
| 3 1 4 3 | 2 |
| 1 2 | 3 |
| 3 3 4 5 | |
| 1 1 | |
| 3 1 2 4 | |

## Problem C. LCA online

| | |
|---|---|
| Input file: | `lca.in` |
| Output file: | `lca.out` |
| Time limit: | 3 seconds |
| Memory limit: | 256 mebibytes |

Of course you all are familiar with the famous LCA problem. Now you are asked to solve its online version.

Let us recall the problem itself. You are given a rooted tree. It means that each vertex except one (that one is called the *root* of the tree) has some other vertex as a parent. Each vertex is a descendant (that is, direct or indirect child) of the root vertex. Vertex $u$ is called an *ancestor* of vertex $v$ if and only if $u$ coincides with $v$, $u$ is the parent of $v$ or $u$ is an ancestor of the parent of $v$. Vertex $w$ is a *common ancestor* of vertices $u$ and $v$ if and only if $w$ is an ancestor of $v$ and $w$ is an ancestor of $u$. Vertex $w$ is the *least common ancestor* (LCA) of vertices $u$ and $v$ if and only if $w$ is a common ancestor of $u$ and $v$ and no other common ancestor of $u$ and $v$ has $w$ as its parent. It can easily be seen that there is only one least common ancestor for any two vertices.

The LCA problem itself is as follows. Given a tree and series of queries (pairs of vertices), find the LCA for each of the given pairs.

You have to solve the online version of this problem. The queries are the same, but now, the tree itself can be changed between queries. For simplicity, we will consider only one type of change: changing the parent of a vertex.

### Input

On the first line of input, there is single integer $n$, the number of vertices in the tree ($1 \le n \le 10^5$). The vertices are conveniently numbered by integers from 1 to $n$. On the second line, there are $(n-1)$ integers $p_2, p_3, \ldots, p_n$. Here, vertex 1 is the root of the tree and each vertex $i > 1$ has vertex $p_i$ as its parent. It is guaranteed that the input data forms a valid rooted tree.

The third line contains one integer $m$, the total number of queries and changes ($1 \le m \le 10^5$). Each of the following $m$ lines contains a description of a query or a tree change request. A query starts with the letter '`Q`' followed by a pair of integers $u$ and $v$, the number of vertices for which you have to find their least common ancestor. A tree change request starts with the letter '`C`' followed by two integers $u$ and $v$ which mean that the new parent of vertex $u$ will be vertex $v$. It is guaranteed that all changes are valid in the sense that, after each change, vertex 1 is still the ancestor of all other vertices.

### Output

For each query, print one integer on a line: the least common ancestor of the specified pair of vertices.

### Examples

| `lca.in` | `lca.out` |
|---|---|
| `5`<br>`1 1 2 4`<br>`3`<br>`Q 3 5`<br>`C 2 3`<br>`Q 3 5` | `1`<br>`3` |
| `3`<br>`1 1`<br>`3`<br>`C 2 3`<br>`C 3 1`<br>`Q 3 1` | `1` |

## Problem D. YAPT

| | |
|---|---|
| Input file: | `palindrome.in` |
| Output file: | `palindrome.out` |
| Time limit: | 2 seconds |
| Memory limit: | 256 mebibytes |

Here is Yet Another Palindrome Task. You are given a string $S$ of length $N$ consisting of lowercase English letters. There are $M$ queries like of the form $(i, k)$. Each query means that you should find the length of $k$-th substring of the given string $S$ which starts from position $i$ of that string and is not a palindrome. The substrings are ordered by their length.

### Input

The first line of input contains the string $S$ of length $N$ ($1 \le N \le 10^5$). This string consists of lowercase English letters.

The second line contains the number of queries $M$ ($1 \le M \le 10^5$) followed by $M$ pairs of integers $(i_j, k_j)$, the queries themselves ($1 \le i_j, k_j \le N$).

### Output

For each query, print a single line containing the length of $k_j$-th substring of the given string $S$ which starts from position $i_j$ of that string and is not a palindrome. If there are less than $k_j$ such strings, print the number $-1$ instead.

### Example

| palindrome.in | palindrome.out |
|---|---|
| <pre>abacaba<br>8<br>1 1<br>1 2<br>1 3<br>4 1<br>4 2<br>4 3<br>4 4<br>2 2</pre> | <pre>2<br>4<br>5<br>2<br>3<br>4<br>-1<br>3</pre> |

## Problem E. Paths

| | |
|---|---|
| Input file: | `paths.in` |
| Output file: | `paths.out` |
| Time limit: | 3 seconds *(7 seconds for Java)* |
| Memory limit: | 256 mebibytes |

The famous mafiosi Gatto Nero and his gang are in trouble. They successfully robbed the Federal Bank of Catland and are now planning the way to escape. There are $N$ cats in the gang and Gatto Nero found $N$ hideouts in different cities for them (he does not need a hideout for himself because he did not take part in the robbery). The gang leader decided that each cat will go to different hideout, so that there will be exactly one cat in each hideout. For safety, the gang selected some roads in the country in such a way that there is exactly one simple path between every pair of hideouts and between the bank and each hideout.

However, there is one problem left: on every road (even on roads selected by the gang) there is a customs official who is of course informed about every member of the gang and will not let them pass. Luckily, the officials are very greedy, so the gang members can *bribe* them: for some amount of money, an official can become "blind" for a short period of time which is enough for one gang member to travel by the road controlled by that official. In order for several gang members to travel by one road, they should all separately bribe the official controlling that road.

Each gang member starts at the bank and moves along the simple path to the hideout assigned to him. For each road the gang member travels by, he has to bribe the official on that road exactly once. Briberies are performed in the order the gang member moves along his path.

Each official $i$ has *rank* $b_i$ and *greediness* $c_i$. The rank denotes importance of the official in the bureaucratic hierarchy. Each official hates all officials with rank less than his own rank. Because of that, an official can be bribed by a gang member only if that official gets strictly more money than at least half of the officials which were already bribed by this gang member and have strictly lower rank. In case there are no such officials, he will be satisfied with the amount of money equal to his greediness instead. A formal definition follows.

Suppose a gang member already bribed a set $U$ of officials and is now trying to bribe official $i$. Let $A \subseteq U$ be the set of officials already bribed by this gang member which have rank strictly lower than $b_i$. If this set $A$ is empty, official $i$ can be bribed for the amount of money equal to his greediness $c_i$. Otherwise, let $m_1 \le m_2 \le \ldots \le m_{|A|}$ be the amounts of money paid to the officials from set $A$ in non-decreasing order. The official $i$ can then be bribed if the amount of money paid to him is strictly greater than $m_k$ where $k = \left\lceil \frac{|A|}{2} \right\rceil$. The amount of money must be an integer. In the latter case, greediness $c_i$ does not matter.

Gatto Nero is tired after the robbery, so he asked you to find how much money will each member of the gang need to spend on bribing officials.

## Input

The first line of input contains the number of gang members $N$ ($1 \le N \le 10^5$). The following $N$ lines contain the descriptions of the roads. Each description consists of four integers $s_i$, $t_i$, $b_i$ and $c_i$, where $s_i$ and $t_i$ are numbers of objects connected by the road (the bank has number 0, hideouts are numbered from 1 to $N$) while $b_i$ and $c_i$ are rank and greediness of the official controlling this road ($0 \le b_i, c_i \le 10^9$). All roads are bidirectional.

## Output

On the first line of output, print $N$ integers: for each hideout from 1 to $N$ in increasing order, print the amount of money which will be spent on bribing officials by the gang member going to this hideout.

## Example

| paths.in | paths.out |
|---|---|
| 8<br>0 1 1 2<br>1 3 0 3<br>1 4 2 4<br>4 7 3 10<br>4 8 0 10<br>0 2 10 4<br>2 5 8 10<br>2 6 11 10 | 2 4 5 5 14 9 8 15 |

## Problem F. Geffoi's Scarf

| | |
|---|---|
| Input file: | `scarf.in` |
| Output file: | `scarf.out` |
| Time limit: | 2 seconds |
| Memory limit: | 256 mebibytes |

Geffoi is a joyful guy. He finds his happiness in simple things that surround him. One day, he decided to buy a scarf that is long enough, and he did it! As Geffoi is a programmer, he bought the scarf which is a stripe divided into $N$ equal pieces with numbers 1, 2, …, $N$ ordered from left to right.

As soon as Geffoi bought the scarf, he needed to fold it. He decided to fold the scarf $K$ times. Each folding step is the following process:

1. Let the length of the packed scarf at this moment be equal to $4M$ for some integer $M$.
2. Divide mentally the scarf into four equal parts numbered from left to right.
3. Lay the first part from above onto the second part by rotating the first part 180 degrees around their common point.
4. Lay the last part from above onto the third part by rotating the last part 180 degrees around their common point.

Note that after each step, the length of the packed scarf is divided by two, and its thickness is increased: the number of layers is doubled. Obviously, before each step, the length of the scarf must be divisible by four.

After $K$ folding steps, each piece of the scarf belongs to some layer. Geffoi is interested to know the exact location of some pieces of the scarf, namely, the layer number and the position within the layer.

### Input

The first line of input contains three positive integers $N$, $K$ and $M$ ($1 \le N \le 10^{18}$, $1 \le K < 59$, $1 \le M \le 10^5$): initial length of the scarf, the number of folding steps and the number of queries. The next line contains $M$ integers $a_i$ ($1 \le a_i \le N$), the numbers of pieces for which Geffoi wants to know the exact location after his packing operation. It is guaranteed that $N$ is divisible by $2^{K+1}$, so that each of the $K$ folding steps is possible.

### Output

For each query, print a single line containing two integers $r_i$ and $c_i$: the number of layer containing the piece in question and the position of the piece within the layer. Layers are numbered from bottom to top starting from 1. Positions in each layer are numbered from left to right starting from 1. See examples for further clarification.

## Examples

| scarf.in | scarf.out |
|---|---|
| 12 1 12 | 2 3 |
| 1 2 3 4 5 6 7 8 9 10 11 12 | 2 2 |
| | 2 1 |
| | 1 1 |
| | 1 2 |
| | 1 3 |
| | 1 4 |
| | 1 5 |
| | 1 6 |
| | 2 6 |
| | 2 5 |
| | 2 4 |
| 8 2 8 | 2 1 |
| 1 2 3 4 5 6 7 8 | 3 1 |
| | 4 1 |
| | 1 1 |
| | 1 2 |
| | 4 2 |
| | 3 2 |
| | 2 2 |

## Problem G. Square

| | |
|---|---|
| Input file: | `square.in` |
| Output file: | `square.out` |
| Time limit: | 2 seconds |
| Memory limit: | 256 mebibytes |

You are given a mirror unit square. A ray of light starts from a light emitter at the bottom left corner of the square in the direction which forms an angle $\alpha$ with the bottom side of the square. There are light receivers at each corner of the square except the bottom left corner. If a ray reaches a light receiver, it will disappear. In case a ray manages to reach the bottom left corner where it originated from, it will also disappear. The square is perfect which means, among other things, that when the ray hits a side of the square in any point except the corners, it is reflected back into the square, and the angle of incidence is equal to the angle of reflection.

You are to count the number of different starting angles $\alpha$ that will result in the ray reaching the light receiver in the upper right corner after exactly $k$ reflections.

### Input

The first line of input contains one integer $k$: the number of reflections ($0 \le k \le 10^{12}$).

### Output

On the first line of output, print one integer: the number of different angles $\alpha$ that will result in the ray reaching the light receiver in the upper right corner after exactly $k$ reflections.

### Examples

| square.in | square.out |
|---|---|
| 0 | 1 |
| 1 | 0 |

# Problem H. Strings

| | |
|---|---|
| Input file: | `strings.in` |
| Output file: | `strings.out` |
| Time limit: | 6 seconds |
| Memory limit: | 256 mebibytes |

This problem features a story of a certain Black Kitten. Perhaps you saw a funny image where a few kittens sit and think about some mundane stuff, like "I want this", "I want that", while the black one thinks "I want to rule the Universe"... Well, that's true! The Black Kitten took a long sheet of paper and wrote a String on it. That String should give him power to rule the Universe!.. *(Or was he just going to win ACM ICPC Finals in 2013 with it?)*

The sad thing is that we'll never know for sure because of other kittens and their stupid games. As you may know, many kittens like to shred furniture, wallpaper and other stuff like that. To make a long story short, a certain White Kitten shred the paper on which the String was written! No more ultimate power for the Black one. *(What a mess.)*

Luckily, the String was only cut into two parts: the First part and the Second one. The Black Kitten now performs some rituals with the two parts. That, he believes, will help him gain some of the magical power. *(Or maybe he is just practicing for the Finals?)*

Anyway, right now, the Black Kitten calculates the number of different substrings of the First part that occur in the Second part exactly $K$ times. Two or more occurrences may overlap. Two substrings are considered different if they are not equal as strings. You may try and calculate the answer for him. Perhaps some of the power will be yours, too. *(At the very least, you have a chance to improve your programming and problem solving skills by doing so.)*

## Input

The first line of input contains the First part of the String. The second line contains the Second part. Each of the parts is a non-empty string consisting of lowercase English letters, and the size of each is no more than 10 000 characters. The third line contains one positive integer $K$ not exceeding the length of each of the given parts.

## Output

The first line of output should contain one integer: the number of different substrings of the First part that occur in the Second part exactly $K$ times.

## Examples

| `strings.in` | `strings.out` |
|---|---|
| `petrozavodsk`<br>`summersession`<br>`4` | `1` |
| `petrozavodsk`<br>`summersession`<br>`1` | `2` |
| `petrozavodsk`<br>`summersession`<br>`2` | `1` |

## Problem I. Triangle

| | |
|---|---|
| Input file: | `triangle.in` |
| Output file: | `triangle.out` |
| Time limit: | 1.5 seconds *(2.5 seconds for Java)* |
| Memory limit: | 256 mebibytes |

You are given a regular polygon consisting of $n$ vertices. In this problem, we will also consider two special cases: a regular 1-gon is a point and a regular 2-gon is a segment. The vertices are painted in two colors: white and black. Your task is to calculate the number of different isosceles triangles such that their vertices coincide with three distinct vertices of the polygon, and these three polygon vertices share the same color. Two triangles are considered different if their sets of vertices differ. Recall that a triangle is isosceles if two of its sides are equal.

### Input

On the first line of input, there is an integer $n$, the number of vertices in the given regular polygon ($1 \le n \le 10^5$). The next line contains the coloring of the vertices: a string consisting of characters '`W`' and '`B`'. Here, '`W`' is a white vertex and '`B`' is a black one. The colors of vertices are given in clockwise order.

### Output

On the first line of output, print the number of different isosceles triangles with vertices coinciding with three distinct polygon vertices of the same color.

### Examples

| `triangle.in` | `triangle.out` |
|---|---|
| `5`<br>`WBWBW` | `1` |
| `3`<br>`WBW` | `0` |
| `7`<br>`WBWBWBB` | `3` |