Greetings From Globussoft

- Given below are 5 Programming questions, you have to solve any 3 out of 5 questions.

- These 5 questions you can attempt in any technology like C/C++, java, .Net, PHP

- To solve these 3 questions you've max. 3 hours.

- While Solving these questions you are not allowed to use any Search Engine like Google, Yahoo, Bing …

All the best for your test

Globussoft

QUESTION - 1

Flavius Josephus and 40 fellow rebels were trapped by the Romans. His companions preferred suicide to surrender, so they decided to form a circle and to kill every third person and to proceed around the circle until no one was left. Josephus was not excited by the idea of killing himself, so he calculated the position to be the last man standing (and then he did not commit suicide since nobody could watch).

We will consider a variant of this "game" where every second person leaves. And of course there will be more than 41 persons, for we now have computers. You have to calculate the safe position. Be careful because we might apply your program to calculate the winner of this contest!

Input Specification

The input contains several test cases. Each specifies a number `n`, denoting the number of persons participating in the game. To make things more difficult, it always has the format `"xyez"` with the following semantics: when `n` is written down in decimal notation, its first digit is `x`, its second digit is `y`, and then follow `z` zeros. Whereas `0<=x,y<=9`, the number of zeros is `0<=z<=6`. You may assume that `n>0`. The last test case is followed by the string `00e0`.

Output Specification

For each test case generate a line containing the position of the person who survives. Assume that the participants have serial numbers from `1` to `n` and that the counting starts with person 1, i.e., the first person leaving is the one with number 2. For example, if there are 5 persons in the circle, counting proceeds as `2, 4, 1, 5` and person `3` is staying alive.

Sample Input

```
05e0
01e1
42e0
66e6
00e0
```

Sample Output

```
3
5
21
64891137
```

QUESTION – 2

Your task is to write a program that performs a simple form of run-length encoding, as described by the rules below.

Any sequence of between 2 to 9 identical characters is encoded by two characters. The first character is the length of the sequence, represented by one of the characters `2` through `9`. The second character is the value of the repeated character. A sequence of more than 9 identical characters is dealt with by first encoding 9 characters, then the remaining ones.

Any sequence of characters that does not contain consecutive repetitions of any characters is represented by a `1` character followed by the sequence of characters, terminated with another `1`. If a `1` appears as part of the sequence, it is escaped with a `1`, thus two `1` characters are output.

Input Specification

The input consists of letters (both upper- and lower-case), digits, spaces, and punctuation. Every line is terminated with a newline character and no other characters appear in the input.

Output Specification

Each line in the input is encoded separately as described above. The newline at the end of each line is not encoded, but is passed directly to the output.

Sample Input

```
AAAAAABCCCC
12344
```

Sample Output

```
6A1B14C
11123124
```

QUESTION – 3

In the following we define the basic terminology of trees. A **tree** is defined inductively: It has a **root** which is either an **external node** (a leaf), or an **internal node** having a sequence of trees as its children. An internal node is also called the **parent** of the roots of its child trees. The **level** of a node in a tree is defined inductively: The root has level `0`, and the level of a node is `1` more than the level of its parent node.

Every internal node of a **binary tree** has precisely two children, its left sub-tree and its right sub-tree. Every internal node of a **labelled binary tree** is additionally marked with a string, its label. A **binary search tree** is a labelled binary tree where every internal node `t` satisfies the following condition: All labels of nodes in the left sub-tree of `t` are less than the label of `t` which is, in turn, less than all labels of nodes in the right sub-tree of `t`. For this condition, we assume lexicographic, i.e., alphabetic order on the strings.

An **inorder traversal** of a tree is defined recursively: A leaf is just visited, and for an internal node first its left sub-tree is traversed inorder, then the node itself is visited, finally its right sub-tree is traversed inorder. It follows that an inorder traversal of a binary search tree yields the labels in lexicographic order. Note that binary search trees whose shapes differ may nevertheless yield the same sequence of strings while being traversed inorder.

When a given string `s` is looked for in a binary search tree, we compare `s` to the label `l` of the root. We are done if $s=l$, otherwise if $s<l$ we continue to search in the left sub-tree, and if $s>l$ in the right sub-tree. If a leaf is reached, we know that `s` is not in the tree.

The number of comparisons performed in such a search procedure depends on `s` and the actual shape of the search tree. Therefore, there is an interest in constructing binary search trees that store a given sequence of strings but provide as efficient access as possible. Of course, we don't know in advance which strings will be looked up in the tree, so we need to make some assumptions.

Let n be the number of strings that are to be stored in the binary search tree. Let $K_1,...,K_n$ be these strings in lexicographic order. Let $p_1,...,p_n$ and $q_0,...,q_n$ be $2n+1$ non-negative real numbers such that $\sum_{i=1..n} p_i + \sum_{i=0..n} q_i = 1$. The interpretation of these numbers is:

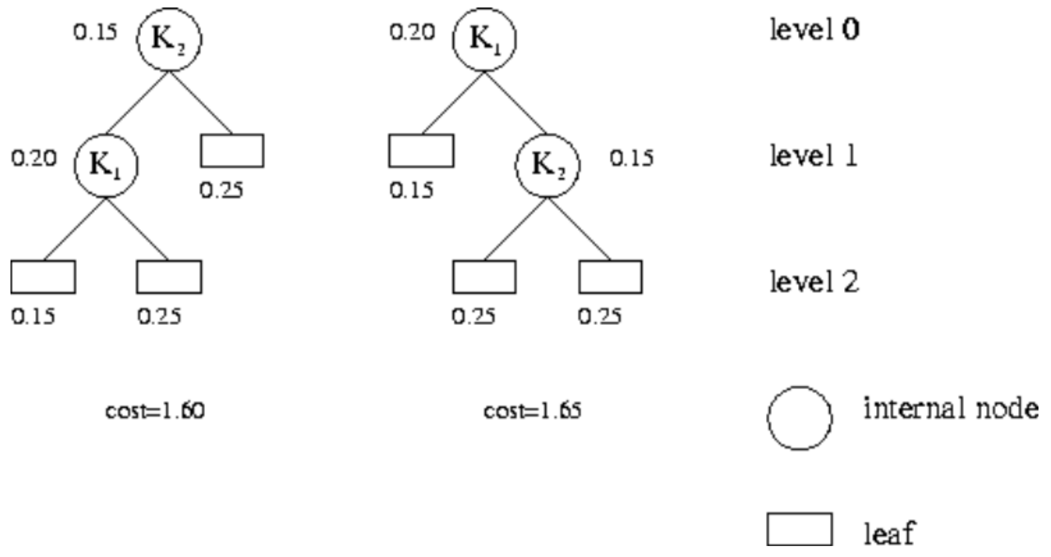
- p_i = probability that the search argument s is K_i.
- q_i = probability that s lies (lexicographically) strictly between K_i and K_{i+1}.

By convention, q_0 is the probability that s is less than K_1, and q_n is the probability that s is greater than K_n. We want to find a binary search tree containing nodes with labels $K_1,...,K_n$ that minimises the expected number of comparisons in the search, namely

$$\text{cost} = \sum_{i=1..n} p_i*(1 + \text{level of internal node } K_i) + \sum_{i=0..n} q_i*(\text{level of leaf between } K_i \text{ and } K_{i+1}).$$

The leaf between K_i and K_{i+1} is that leaf reached in the search for a string s that lies (lexicographically) strictly between K_i and K_{i+1}. Adhere to the convention stated above for the border cases.

The following figure illustrates the first test case of the sample input. It shows the two possible binary search trees, the probabilities and the associated costs.



Input Specification

The input contains several test cases. Every test case starts with an integer `n`. You may assume that `1<=n<=200`. Then follow `2n+1` non-negative integers denoting frequencies. Let `s` be the sum of all frequencies. You may assume that `1<=s<=1000000`. The probabilities $p_1, \ldots, p_n$ and $q_0, \ldots, q_n$ are calculated in this order by dividing the frequencies by `s`. The last test case is followed by a zero.

Output Specification

For each test case devise a binary search tree whose cost is minimal for the specified probabilities. Output the integer `cost*s` for such a tree.

Sample Input

```
2
20 15 15 25 25
35
142 35 58 5 20 5 10 9 15 23 129 4 52 5 38 18 9 7 2 4 266 93 5 18 18 27 5 10 11 180 4 32 21 3 21
0 55 27 36 85 31 58 3 334 0 98 27 113 89 180 0 62 12 0 37 0 3 64 70 0 277 0 0 0 170 0 18 76 27 3 29
0
```

Sample Output

```
160
13637
```

QUESTION – 4

After the 1997/1998 Southwestern European Regional Contest (which was held in Ulm) a large contest party took place. The organization team invented a special mode of choosing those participants that were to assist with washing the dirty dishes. The contestants would line up in a queue, one behind the other. Each contestant got a number starting with 2 for the first one, 3 for the second one, 4 for the third one, and so on, consecutively.

The first contestant in the queue was asked for his number (which was 2). He was freed from the washing up and could party on, but every second contestant behind him had to go to the kitchen (those with numbers 4, 6, 8, etc). Then the next contestant in the remaining queue had to tell his number. He answered 3 and was freed from assisting, but every third contestant behind him was to help (those with numbers 9, 15, 21, etc). The next in the remaining queue had number 5 and was free, but every fifth contestant behind him was selected (those with numbers 19, 35, 49, etc). The next had number 7 and was free, but every seventh behind him had to assist, and so on.

Let us call the number of a contestant who does not need to assist with washing up a lucky number. Continuing the selection scheme, the lucky numbers are the ordered sequence 2, 3, 5, 7, 11, 13, 17, etc. Find out the lucky numbers to be prepared for the next contest party.

Input Specification

The input contains several test cases. Each test case consists of an integer `n`. You may assume that `1<=n<=3000`. A zero follows the input for the last test case.

Output Specification

For each test case specified by `n` output on a single line the `n`-th lucky number.

Sample Input

```
1
2
10
20
0
```

Sample Output

```
2
3
29
83
```

QUESTION – 5

A set of laboratory mice is being trained to escape a maze. The maze is made up of cells, and each cell is connected to some other cells. However, there are obstacles in the passage between cells and therefore there is a time penalty to overcome the passage Also, some passages allow mice to go one-way, but not the other way round.

Suppose that all mice are now trained and, when placed in an arbitrary cell in the maze, take a path that leads them to the exit cell in minimum time.

We are going to conduct the following experiment: a mouse is placed in each cell of the maze and a count-down timer is started. When the timer stops we count the number of mice out of the maze.

Problem

Write a program that, given a description of the maze and the time limit, predicts the number of mice that will exit the maze. Assume that there are no bottlenecks is the maze, i.e. that all cells have room for an arbitrary number of mice.

Input

The maze cells are numbered $1, 2, \ldots, N$, where N is the total number of cells. You can assume that $N \leq 100$.

The first three input lines contain N, the number of cells in the maze, E, the number of the exit cell, and the starting value T for the count-down timer (in some arbitrary time unit).

The fourth line contains the number M of connections in the maze, and is followed by M lines, each specifying a connection with three integer numbers: two cell numbers a and b (in the range $1, \ldots, N$) and the number of time units it takes to travel from a to b.

Notice that each connection is one-way, i.e., the mice can't travel from b to a unless there is another line specifying that passage. Notice also that the time required to travel in each direction might be different.

Output

The output consists of a single line with the number of mice that reached the exit cell E in at most T time units.

Example

Input:

```
4
2
1
8
1 2 1
1 3 1
2 1 1
2 4 1
3 1 1
3 4 1
4 2 1
4 3 1
```

Output:

```
3
```