2^o Treino dos Alunos da UFPR

- Este caderno contém 6 problemas e 9 páginas;
- Todos os programas devem ler dados da entrada padrão (stdin) e escrever dados na saída padrão (stdout);
- Cada problema será testado com um único arquivo de entrada. Preste atenção em como este arquivo termina, em cada problema;
- Programadores de **Pascal**: Lembre-se que o tipo *integer* tem apenas 16 bits. O tipo *longint* tem 32 bits e o tipo *int64* tem 64 bits.

Problem A: Windows

File: `windows.[c|cpp|java|pas]`

Emma is not very tidy with the desktop of her computer. She has the habit of opening windows on the screen and then not closing the application that created them. The result, of course, is a very cluttered desktop with some windows just peeking out from behind others and some completely hidden. Given that Emma doesn't log off for days, this is a formidable mess. Your job is to determine which window (if any) gets selected when Emma clicks on a certain position of the screen.

Emma's screen has a resolution of 10^6 by 10^6. When each window opens its position is given by the upper-left-hand corner, its width, and its height. (Assume position (0,0) is the location of the pixel in the upper-left-hand corner of her desktop. So, the lower-right-hand pixel has location (999999, 999999).)

Input

Input for each test case is a sequence of desktop descriptions. Each description consists of a line containing a positive integer n, the number of windows, followed by n lines, $n \leq 100$, describing windows in the order in which Emma opened them, followed by a line containing a positive integer m, the number of queries, followed by m lines, each describing a query. Each of the n window description lines contains four integers r, c, w, and h, where (r, c) is the row and column of the upper left pixel of the window, $0 \leq r, c \leq 999999$, and w and h are the width and height of the window, in pixels, $1 \leq w, h$. All windows will lie entirely on the desktop (i.e., no cropping). Each of the m query description lines contains two integers cr and cc, the row and column number of the location (which will be on the desktop). The last test case is followed by a line containing 0.

Output

Using the format shown in the sample, for each test case, print the desktop number, beginning with 1, followed by m lines, one per query. The i-th line should say either *windowk*, where k is the number of the window clicked on, or *background* if the query hit none of the windows. We assume that windows are numbered consecutively in the order in which Emma opened them, beginning with 1. Note that querying a window does not bring that window to the foreground on the screen.

Sample Input

```
3
1 2 3 3
2 3 2 2
3 4 2 2
4
3 5
1 2
4 2
3 3
2
5 10 2 10
100 100 100 100
2
5 13
100 101
0
```

Sample Output

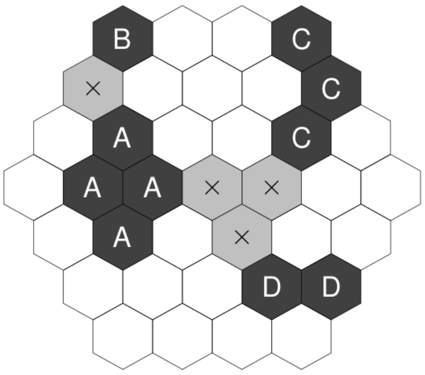
```
Desktop 1:
window 3
window 1
background
window 2
Desktop 2:
background
window 2
```

Problem B: Hexagonal Parcels

File: `hexagonal.[c|cpp|java|pas]`

A civil engineer that has recently graduated from the Czech Technical University encountered an interesting problem and asked us for a help. The problem is more of economical than engineering nature. The engineer needs to connect several buildings with an infrastructure. Unfortunately, the investor is not the owner of all the land between these places. Therefore, some properties have to be bought first.

The land is divided into a regular "grid" of hexagonal parcels, each of them forms an independent unit and has the same value. Some of the parcels belong to the investor. These parcels form four connected areas, each containing one building to be connected with the others. Your task is to find the minimal number of parcels that must be acquired to connect the four given areas.



The whole land also has a hexagonal shape with six sides, each consisting of exactly H parcels. The above picture shows a land with $H = 4$, parcels with letters represent the four areas to be connected. In this case, it is necessary to buy four additional parcels. One of the possible solutions is marked by crosses.

Input

The input contains several scenarios. Each scenario begins with an integer number H, which specifies the size of the land, $2 \leq H \leq 20$. Then there are $2 * H - 1$ lines representing individual "rows" of the land (always oriented as in the picture). The lines contain one non-space character for each parcel. It means the first line will contain H characters, the second line $H + 1$, and so on. The longest line will be the middle one, with $2 * H - 1$ characters. Then the "length" descends and the last line contains H parcels, again.

The character representing a parcel will be either a dot (".") for the land that is not owned by the investor, or one of the uppercase letters "A", "B", "C", or "D". The areas of parcels occupied by the same letter will always be connected. It means that between any two parcels in the same area, there exists a path leading only through that area.

Beside the characters representing parcels, the lines may contain any number of spaces at any positions to improve "human readability" of the input. There is always at least one space between two letters (or the dots). After the land description, there will be one empty line and then the next scenario begins. The last scenario is followed by a line containing zero.

Output

For each scenario, output one line with the sentence "You have to buy P parcels.", where P is the minimal number of parcels that must be acquired to make all four areas connected together.

Areas are considered connected, if it is possible to find a path between them that leads only through parcels that have been bought.

Sample Input

```
4
    B . . C
   . . . . C
  . A . . C .
 . A A . . . .
  . A . . . .
   . . . D D
    . . . .

0
```

Sample Output

```
You have to buy 4 parcels.
```

Problem C: Anagrama

`File: anagrama.[c|cpp|java|pas]`

Uma string x é um anagrama da string y se podemos reordenar as letras da string x e formar exatamente a string y. Por exemplo, a string "DOG" é um anagrama de "GOD", assim como "BABA" é anagrama de "AABB". A string "ABBAC" não é um anagrama de "CAABA".

São dadas duas strings s e t, que têm o mesmo tamanho e contém apenas letras maiúsculas do alfabeto inglês (A,B,C, ..., X, Y, Z). Através da aplicação sucessiva de operações, você deve transformar a string s em algum anagrama da string t.

Uma operação sobre uma string consiste em trocar qualquer (mas apenas uma) letra da string por qualquer outra letra do alfabeto inglês (A, ..., Z).

Transforme a string s em um anagrama da string t com o menor número possível de operações. Se você consegue obter vários anagramas de t com o mínimo de operações, obtenha o lexicograficamente menor. A ordem lexicográfica de strings é a conhecida ordem "de dicionário", ou "alfabética".

Input

Cada caso de teste consiste em duas linhas. A primeira linha contém a string s, e a segunda contém a string t. As strings têm o mesmo tamanho (de 1 a 10^5 caracteres) e consiste apenas de letras maiúsculas (A-Z). O último caso de teste é seguido por duas linhas contendo #.

Output

Para cada caso de teste, imprima duas linhas. Na primeira, imprima z - o número mínino de operações a serem aplicadas sobre s para se obter um anagrama de t. Na segunda linha, imprima o anagrama obtido com z operações. Lembre-se que o anamagra obtivo deve ser o menor possível, lexicograficamente.

Sample Input

```
ABA
CBA
CDBABC
ADCABD
AABAA
BBAAA
#
#
```

Sample Output

```
1
ABC
2
ADBADC
1
AABAB
```

Problem D: Hailstone Sequences

File: hailstone.[c|cpp|java|pas]

Consider the sequence formed by starting from a positive integer h_0 and iterating with $n = 1, 2, ...$ the following definition until $h_n = 1$:

$$h_n = \begin{cases} \frac{1}{2} \times h_{n-1} & \text{if } h_{n-1} \text{ is even;} \\ \\ 3 \times h_{n-1} + 1 & \text{if } h_{n-1} \text{ is odd.} \end{cases}$$

For instance, if we start with $h_0 = 5$ the following sequence is generated: $5, 16, 8, 4, 2, 1$. If we start with $h_0 = 11$, the sequence generated is $11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1$. As you can see from these examples, the numbers go up and down, but eventually comes down to end in 1 (at least for all numbers that have ever been tried). These sequences are called Hailstone sequences because they are similar to the formation of hailstones, which get carried upward by the winds over and over again before they finally descend to the ground. In this problem, given a positive integer, your task is to compute the highest number in the Hailstone sequence which starts with the given number.

Input

Each test case is described using a single line. The line contains an integer H representing the starting value to build the sequence ($1 \leq H \leq 500$). The last test case is followed by a line containing one zero.

Output

For each test case output a line with an integer representing the highest number in the Hailstone sequence that starts with the given input value.

Sample Input

```
5
11
27
0
```

Sample Output

```
16
52
9232
```

Problem E: Lucky Substring

`File: lucky.[c|cpp|java|pas]`

Petya loves lucky numbers. Everybody knows that lucky numbers are positive integers whose decimal representation contains only the lucky digits 4 and 7. For example, numbers 47, 744, 4 are lucky and 5, 17, 467 are not.

One day Petya was delivered a string s, containing only digits. He needs to find a string that:

- represents a lucky number without leading zeroes,
- is not empty,
- is contained in s as a substring the maximum number of times.

Among all the strings for which the three conditions given above are fulfilled, Petya only needs the lexicographically minimum one. Find this string for Petya.

Input

The input contains several test cases. Each test case contains a single line with a non-empty string s whose length can range from 1 to 50, inclusive. The string only contains digits. The string can contain leading zeroes. The last test case is followed by a line containing a single zero. There will not be any test case consisting of only one zero.

Output

For each test case print one line with the answer to Petya's problem. If the sought string does not exist, print "-1" (without quotes).

Sample Input

```
047
16
472747
0
```

Sample Output

```
4
-1
7
```

The lexicographical comparison of strings is performed by the $\leq$ operator in the modern programming languages. String x is lexicographically less than string y either if x is a prefix of y, or exists such $i(1 \leq i \leq min(|x|, |y|))$, that $x_i \leq y_i$ and for any $j(1 \leq j \leq i)x_j = y_j$. Here $|a|$ denotes the length of string a.

In the first sample three conditions are fulfilled for strings "4", "7" and "47". The lexicographically minimum one is "4".

In the second sample s has no substrings which are lucky numbers.

In the third sample the three conditions are only fulfilled for string "7".

Problem F: Isosceles Triangles

`File: triangles.[c|cpp|java|pas]`

A given triangle can be either equilateral (three sides of the same length), scalene (three sides of different lengths), or isosceles (two sides of the same length and a third side of a different length). It is a known fact that points with all integer coordinates cannot be the vertices of an equilateral triangle.

You are given a set of different points with integer coordinates on the XY plane, such that no three points in the set lay on the same line. Your job is to calculate how many of the possible choices of three points are the vertices of an isosceles triangle.

Input

There are several test cases. Each test case is given in several lines. The first line of each test case contains an integer N indicating the number of points in the set ($3 \leq N \leq 1000$). Each of the next N lines describes a different point of the set using two integers X and Y separated by a single space ($1 \leq X, Y \leq 10^6$); these values represent the coordinates of the point on the XY plane. You may assume that within each test case no two points have the same location and no three points are collinear.

The last test case is followed by a line containing a single zero.

Output

For each test case output a single line with a single integer indicating the number of subsets of three points that are the vertices of an isosceles triangle.

Sample Input

```
5
1 2
2 1
2 2
1 1
1000 1000000
6
1000 1000
996 1003
996 997
1003 996
1003 1004
992 1000
0
```

Sample Output

```
4
10
```