Welcome to ACM/ICPC Asia Kaohsiung!

1. There are ten problems in this programming contest.
2. Each problem has its own time limit which is shown in the problem.
3. The input file of your program for each problem must be `stdin` in C/C++. The input file of your program will be redirected to the test data file used by the judge.
4. The output of your program should be `stdout`. It will be redirected to a file and then shown on the screen.

Problem A
Spread Out Message
Time Limit: 2 seconds

Wakkas is a great chief of a tribe. As a leader, every day he has to deal with many jobs. Once he makes a decision, the message must be sent to all of his people. Traditionally, Wakkas announces a message in a square and simply let the message be spread out among people. Sometimes, not all people are notified of a message by this way. If a person is not in the square during the announcement of a message, it is possible that no one tells him the message. Even though he hears the message, he may doubt the credibility if the message is sent by a stranger. Another problem is that one person might hear the same message over and over again. Thus, some people do not spread out any message since they are bored to do so.

Recently, Wakkas figures out a more efficient way to spread out a message. He maintains a list, called the *spreading list*, which consists of pairs of persons. Each pair (p_i, p_j) of the list indicates that p_i needs to tell p_j any new message he received and p_j also needs to tell p_i any new message he received. For example, assume that there are four persons p_1, p_2, p_3, and p_4 and $((p_2, p_3), (p_1, p_3), (p_3, p_4))$ is the spreading list. If Wakkas tells a new message to p_1; p_1 in turn tells the message to p_3; and finally, p_3 tells the message to both p_2 and p_4. (See Figure 1.) To announce a new message, Wakkas always tells the message only to p_1.

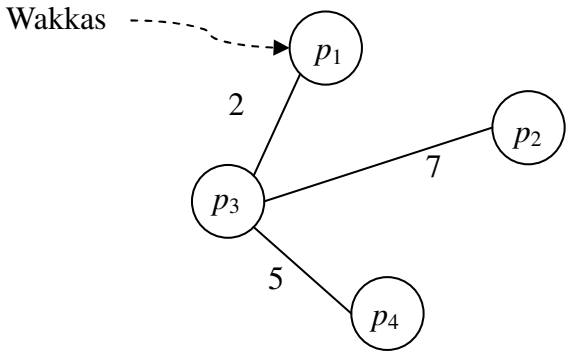
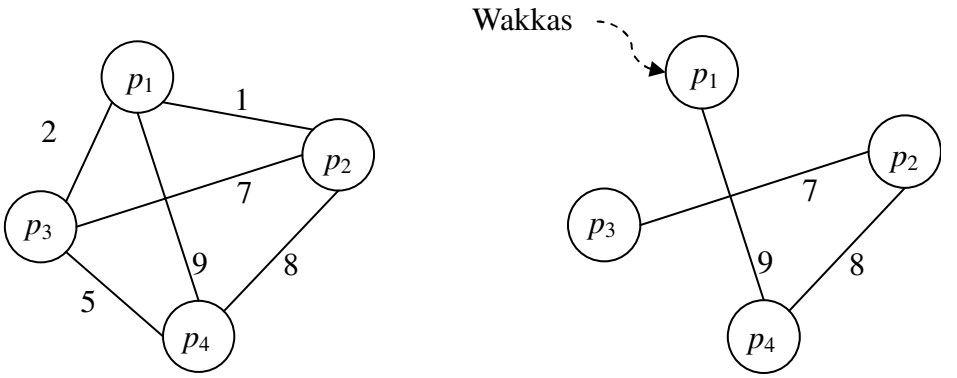


Figure 1. A spreading list $((p_2, p_3), (p_1, p_3), (p_3, p_4))$.

Wakkas wants the length of the spreading list to be as short as possible. After some study, he finds that a list of $n - 1$ pairs is necessary and sufficient for all people to receive a new message, where n is the number of his people. Wakkas also wants that a new message can be received by all of his people as soon as possible. To do so, for every pair (p_i, p_j) of his people, he assigns a *score* $f_{(i,j)}$ according to the degree of familiarity between p_i and p_j, where a higher score indicates that the two people are more familiar with each other. Wakkas defines the score of a spreading list as the total score of its pairs. For example, if $f_{(2,3)} = 7$, $f_{(1,3)} = 2$, and $f_{(3,4)} = 5$, the score of the spreading list in Figure 1 is 14. Wakkas believes that the higher the score of a spreading list, the shorter the time required for all people to receive a new message.

Wakkas' target is to create a spreading list of highest score. The created spreading list must contain exactly $n - 1$ pairs which are sufficient for all people to receive a new message.

Consider the example in Figure 2(a). Figure 2(b) shows a spreading list with the highest score 24. Wakkas is too busy to handle it; therefore, he seeks for your help.



(a) scores $f_{(i,j)}$ (b) a spreading list with score 24

Figure 2. An example of $n = 4$.

Technical Specification

1. The number of people in the tribe, $n : 2 \leq n \leq 100$.
2. The score, $f_{(i,j)} : 0 \leq f_{(i,j)} \leq 100$.

Input

There are at most 100 test cases. Each test case describes the tribe in two parts. The first part is a single integer n, where $2 \leq n \leq 100$. The second part consists of n lines indicating the scores among the n people. The i^{th} line contains n integers $f_{(i,1)}, f_{(i,2)}, ..., f_{(i,n)}$. Note that $f_{(i,j)} = f_{(j,i)}$ for $1 \leq i, j \leq n$, and $f_{(i,i)} = 0$ for $1 \leq i \leq n$.

The last test case will be followed by a line consisting of an integer 0.

Output

For each test case, print a line containing the highest score of a spreading list.

Sample Input

```
3
0 5 2
5 0 3
2 3 0
```

```
4
0 1 2 9
1 0 7 8
2 7 0 5
9 8 5 0
0
```

Sample Output

```
8
24
```

Problem B
Bricks
Time Limit: 1 seconds

Patrick is a big fan of the bricks. He has been collecting the bricks of different colors, sizes, and shapes, since he was little. Recently, he got a job in another city, and he planned to move next week. Since the new apartment is much smaller than the old one, Patrick decided to give away his bricks to the kids in his neighborhood. However, to receive the gift, the kid must play the bricks with Patrick and solve a brick game. The game is quite simple and the rules are as follows:

1. The kid chooses a number N by casting lots.
2. Then, Patrick randomly selects N bricks from his collection.
3. The kid has to figure out the *similar* bricks from the N ones.
4. The kid will obtain the N bricks for free if he gives the correct answer.

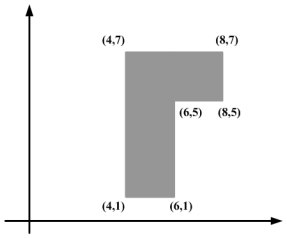
Note that the two bricks are considered *similar* if and only if *their corresponding angles are all congruent and corresponding sides are all proportional with/without 180-degree flipping.*

For the ease of representation, we describe each brick by a sequence of vertices along the border of the brick in clockwise order using the 2D Cartesian coordinate system. For instance, if $N = 4$, and the four bricks are $B_1 = \{(4,1)\ (4,7)\ (8,7)\ (8,5)\ (6,5)\ (6,1)\}$, $B_2 = \{(6,5)\ (6,3)\ (5,3)\ (5,4)\ (3,4)\ (3,5)\}$, $B_3 = \{(10,1)\ (1,1)\ (1,4)\ (7,4)\ (7,7)\ (10,7)\}$, and $B_4 = \{(6,3)\ (6,1)\ (4,1)\ (4,7)\ (8,7)\ (8,3)\}$. We know that B_1 and B_2 are similar, because their corresponding angles are all congruent and corresponding sides are all proportional. Moreover, B_2 and B_3 are similar, because, after flipping B_3, their corresponding angles are all congruent and corresponding sides are all proportional. Meanwhile, B_4 is not similar to any other bricks. Thus, we know the answer of this brick game is 3, and the three similar bricks are B_1, B_2, and B_3.

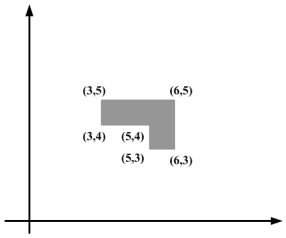
For simplicity, we assume N is an integer, and $3 \leq N \leq 10$. Each polygon is simple (i.e., the boundary of the polygon does not cross itself) and represented by a list of P vertices in clockwise order. P is an integer, and $3 \leq P \leq 10$. The X, Y coordinates of the i-th vertex of the polygon is (X_i, Y_i). X_i and Y_i are positive real numbers to four decimal places; moreover, $0 < X_i \leq 1,000$ and $0 < Y_i \leq 1,000$. Finally, for each brick game, there is either none or ***only one*** set of bricks that are similar. In addition, all bricks in a game have the same number of vertices.

Input Format

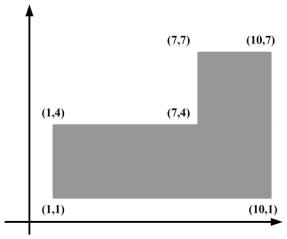
There are multiple test cases in the input file. For each test case, the first line contains a positive integer N representing the number of polygons in the case, and $N = 0$ indicates the end of the test data. The second line contains a positive integer P, representing the number of vertices contained by each polygon. In the following N lines, the i-th line contains $2P$ positive real numbers to four decimal places, and they are separated by one single space. Specifically, the $(2k-1)$-th and $(2k)$-th real numbers in the i-th line represent the X and Y coordinate values of the k-th vertex of the i-th brick respectively.



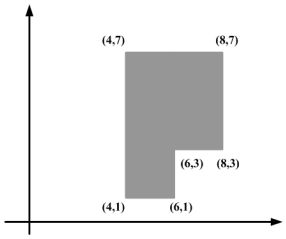
B_1



B_2



B_3



B_4

Output Format

Please output the answer of each test case in two lines. The first line contains an integer, representing the number of similar bricks in the test case. The second line contains the sequence numbers of the bricks that are similar in ascending order and separated by one single space. If there are no similar bricks in the test case, please output 0 in both the first line and the second line.

Sample Input

```
4
6
4.0000 1.0000 4.0000 7.0000 8.0000 7.0000 8.0000 5.0000 6.0000 5.0000 6.0000 1.0000
6.0000 5.0000 6.0000 3.0000 5.0000 3.0000 5.0000 4.0000 3.0000 4.0000 3.0000 5.0000
10.0000 1.0000 1.0000 1.0000 1.0000 4.0000 7.0000 4.0000 7.0000 7.0000 10.0000 7.0000
6.0000 3.0000 6.0000 1.0000 4.0000 1.0000 4.0000 7.0000 8.0000 7.0000 8.0000 3.0000
4
3
```

```
2.0000 1.0000 4.0000 4.0000 6.0000 1.0000
4.0000 7.0000 6.0000 9.0000 11.0000 1.0000
6.0000 9.0000 12.0000 6.0000 6.0000 6.0000
4.0000 13.0000 4.0000 1.0000 2.0000 7.0000
4
3
2.0000 1.0000 4.0000 4.0000 6.0000 1.0000
4.0000 7.0000 6.0000 9.0000 11.0000 1.0000
7.0000 2.0000 7.0000 6.0000 10.0000 4.0000
4.0000 13.0000 4.0000 1.0000 2.0000 7.0000
0
```

Sample Output for the Sample Input

```
3
1 2 3
0
0
2
1 3
```

Problem C
2D Folding Game
Time Limit: 5 seconds

Nintendo Game Company develops a game played on a *two-dimensional square lattice* $\Re$ (*2D-lattice*). A 2D-lattice can be viewed as a graph with vertex set $V = \{[a, b] |\ a, b \in \mathbf{Z}\}$. The vertices are represented as ordered pairs and two vertices are *adjacent* if and only if the corresponding squares share a side of lattice. Figure 1 illustrates a 2D-lattice. Given a sequence of length m over the symbols 0 and 1, the conformations of the sequence are restricted to self-avoiding paths embedded in a 2D-lattice, where a *self-avoiding path* is a path in a 2D-lattice that does not visit the same vertex more than once. The detailed description of the game is as follows.

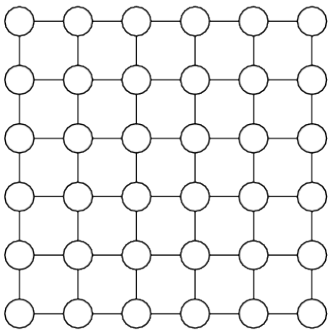


Figure 1: A 2D-lattice

Let $p = p_1 \ldots p_m$ be a sequence of length m over symbols 0 and 1, and let $\Re = (V, E)$ be a 2D-lattice. An *embedding* of p into $\Re$ is an injective function $\phi : \{1, \ldots, m\} \to V$ from the positions of the sequence to the vertices of the lattice that assigns adjacent positions in p to adjacent vertices in $\Re$, i.e., $(\phi(i), \phi(i+1)) \in E$ for all $1 \leq i \leq m-1$. These edges $(\phi(i), \phi(i+1)) \in E$ for $1 \leq i \leq m-1$ are called *binding edges*. An embedding of p into $\Re$ is called a *conformation*. An edge (x, y) of $\Re$ is called *contact edge* if it is not a binding edge, but there exist $i, j \in \{1, \cdots, m\}$ such that $\phi(i) = x, \phi(j) = y$, and $p_i = p_j = 1$.

The *score* of a conformation ϕ equals the number of the contact edges in $\Re$. Figure 2 illustrates a conformation with score 7. An *optimal conformation* of a sequence in $\Re$ is a conformation that maximizes the number of contact edges. Given a sequence, a player is asked to find an optimal conformation and your task is to write a computer program to compute the score of an optimal conformation.

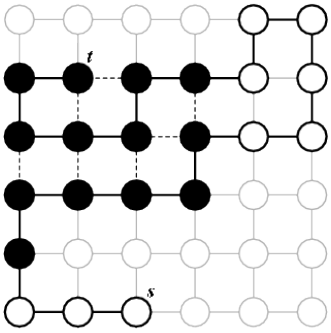


Figure 2: A conformation of the sequence 000111111000000011111111, where " • " represents 1, "∘" represents 0, the bold edges represent the binding edges, and the dotted edges represent the contact edges. The score of this conformation equals 7. In addition, s and t are the beginning 0 and the ending 1 of the sequence.

Technical Specification

$1 \leq m \leq 20$

Input Format

For each test case, there are two lines: the first line contains a positive integer $1 \leq m \leq 20$, where m is the number of symbols in the given sequence and the second line represents the sequence of length m. The next test case is separated from the previous test case by one empty line. A test case that starts with $m = 0$ denotes the end of the input file.

Output Format

For each test case, output the score on one line.

Sample Input

```
3
000
```

```
4
1111

7
1001001

0
```

Sample Output for the Sample Input

```
0
1
2
```

Problem D
Mobile Communications
Time Limit: 15 seconds

Ace Consumer Mobiles (ACM) is currently receiving complaints about the quality of their mobile phone service in some locations of the city. Hence, your team needs to resolve the problem. As a starting point, your boss decides to check whether the current locations of the mobile base stations would cause some problems.

The base stations of ACM works as follows. Each base station a operates with signal of a fixed power level P_a. The physical laws state that the strength of the signal would then decrease proportional to the squared distance between the user and the base station. That is, if the user is of Euclidean distance d to some base station a, the signal strength that the user's phone can sense from the base station is $Q_a = \frac{P_a}{d^2}$.

After several days of checking, you have an important finding. "Sir, I think I found the reason. All the locations that raised complaints share one important property: competing signals between two or more base stations."

"Very good", your boss says. It is a known problem with this transmission protocol, then. Let's say that a phone senses signals of strength Q_a and Q_b from base stations a and b. If both Q_a and Q_b are no less than some level ϵ, the two signals would compete with each other, making the mobile phone unable to function perfectly.

The ideal solution, then, is to decrease the power level of some of the base stations. But there are too many base stations to be tuned individually. Thus, your boss decides to try an approximate solution. "Can we set the P_a of every base station a to a constant $\hat{P}$ to avoid the hazard of competing signals?"

Now, it is you, the engineer, who has to answer this question from your boss.

Input Format

The input file contains a set of test data. The first line of each set is the number of base stations N, which is between 2 and 40000. Each of the next N lines contains two integers x_n and y_n, which denotes the position (x_n, y_n) of the n-th base station. Both x_n and y_n would be between 1 and 1000; the positions of any two base stations would be different. Then, the last line of the set contains two numbers, an integer E that equals $(\epsilon \cdot 1000)$, and an integer $\hat{P}$. E will be between 0 and 1000000 while $\hat{P}$ is between 1 and 1000. A test set that starts with $N = 0$ denotes the end of the test data.

Output Format

For each test set with $N > 0$, output a line of three characters YES if using $\hat{P}$ resolves the problem, or a line of two characters NO if not.

Sample Input

```
2
3 1
1 3
4000 4
```

```
3
3 1
1 3
3 3
4000 4
0
```

Sample Output for the Sample Input

```
YES
NO
```

Problem E
AR Game
Time Limit: 2 seconds

Gamebox is developing a new augmented reality game for young kids. Players interact with the virtual object on screen through movements of markers. A marker is a square playing card with special marking. Markers all have different markings. The game is played by showing a marker to the game console. The game console has a hidden video camera that can automatically locate and take a snapshot of the marker being played. Depending on how the card is being held and the distance from the game console, the captured image of the marker maybe displaced from the upright position. Furthermore, noise may be introduced during the imaging process. The marker is then recognized and the game is played accordingly. Obviously, if the marker were incorrectly recognized, then the game would not be played correctly. You are hired as a game programmer to make sure that the markers are recognized correctly.

Constraints

1. All the images are binary images (containing only two colors: black and white). 0 denotes black pixel and 1 denotes white pixel.
2. There are 5 markers. Each has been digitized into a 100x100 image. The images are shown and described below in graphical form.
3. The captured image (test images) can be
   - Rotated: in 0, 90, 180, or 270 degree angle.
   - Enlarged/shrunk: image size can be of size 50x50, 100x100, 150x150, or 200x200.
   - Contains random noise: at most 3% of the pixels in the input image are reverted. Black becomes white or vise versa.
   - Any combination of the above conditions.

Input Format

The input file contains a set of test images. Each test image is defined by a single integer n on one line to indicate the image size (nxn). This number is either 0, 50, 100, 150, or 200. 0 indicates the end of the test data. This is followed by n rows of n consecutive 0s and 1s, which defines the captured image to be processed.

Output Format

For each input case, output a single integer on one line indicating the marker that matches with the given input image.

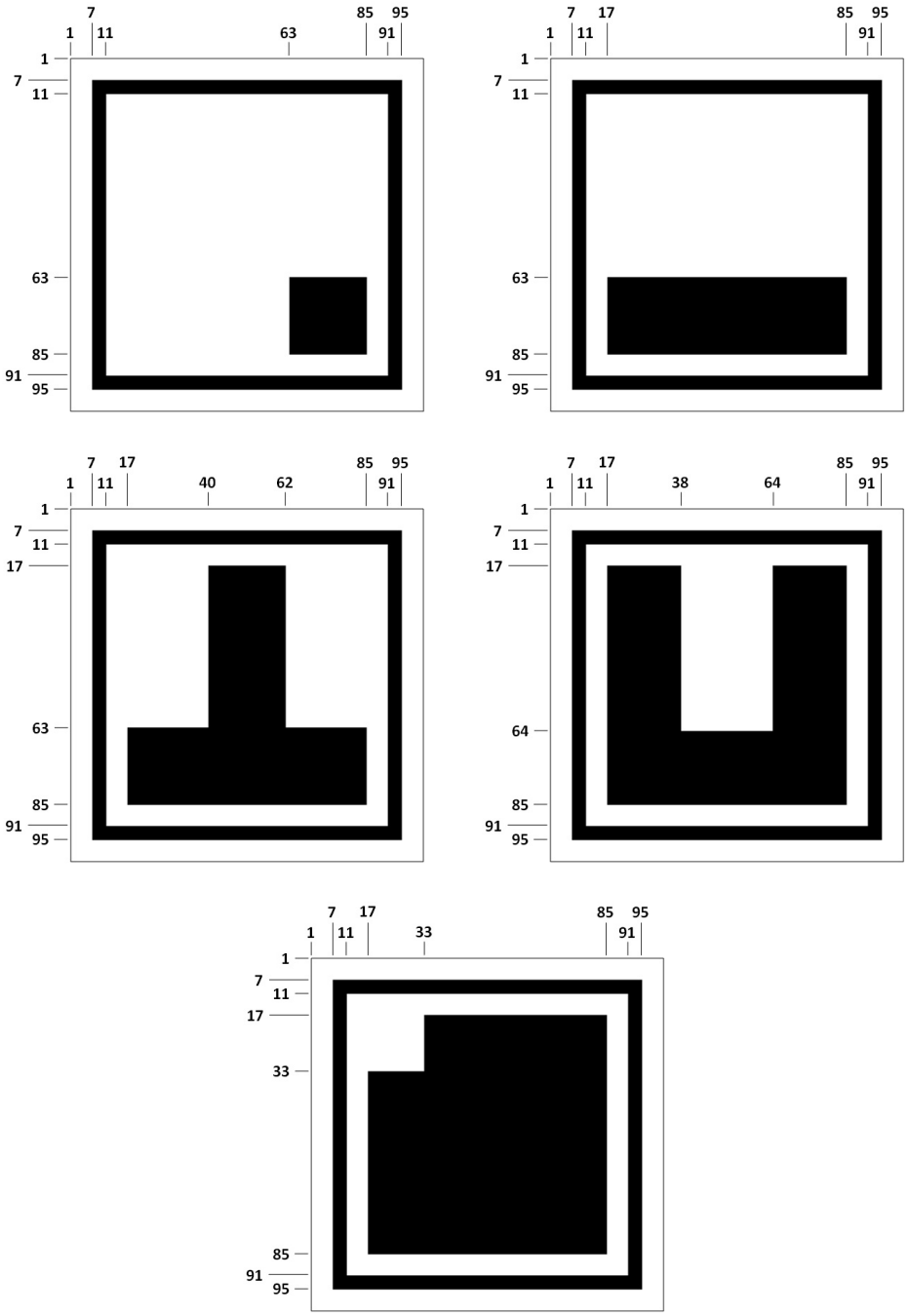


Figure 3: **Markers 1, 2, 3, 4, and 5 from left to right.**

Sample Input

```
50
11111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111
11100000000000000000000000000000000000000000000111
11100000000000000000000000000000000000000000000111
11100111111111111111111111111111111111111111100111
11100111111111111111111111111111111111111111100111
11100111111111111111111111111111111111111111100111
11100111111111111111111111111111000000000011100111
11100111111111111111111111111111000000000011100111
11100111111111111111111111111111000000000011100111
11100111111111111111111111111111000000000011100111
11100111111111111111111111111111000000000011100111
11100111111111111111111111111111000000000011100111
11100111111111111111111111111111000000000011100111
11100111111111111111111111111111000000000011100111
11100111111111111111111111111111000000000011100111
11100111111111111111111111111111000000000011100111
11100111000000000000000000000000000000000011100111
11100111000000000000000000000000000000000011100111
11100111000000000000000000000000000000000011100111
11100111000000000000000000000000000000000011100111
11100111000000000000000000000000000000000011100111
11100111000000000000000000000000000000000011100111
11100111000000000000000000000000000000000011100111
11100111000000000000000000000000000000000011100111
11100111000000000000000000000000000000000011100111
11100111111111111111111111111111000000000011100111
11100111111111111111111111111111000000000011100111
11100111111111111111111111111111000000000011100111
11100111111111111111111111111111000000000011100111
11100111111111111111111111111111000000000011100111
11100111111111111111111111111111000000000011100111
11100111111111111111111111111111000000000011100111
11100111111111111111111111111111000000000011100111
11100111111111111111111111111111111111111111100111
11100111111111111111111111111111111111111111100111
11100111111111111111111111111111111111111111100111
11100000000000000000000000000000000000000000000111
11100000000000000000000000000000000000000000000111
```

```
11111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111
50
11111111111111111111110111111111111111111111110111
11111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111
11100000000000000001000000000000000000000000000111
11101000000000000000000000000000000000000000000111
11100111111111111111111111101111111111111111100111
11100111111111111111111111111111111101111111101111
11100111111111111111111111111111111110111111100111
11100111000000000000000000000000001111111111100111
11100111000000000000000000000001000111110111100110
11100111000000000000000000000000001111111111100111
11100111000001000000000000000000001111111111100111
11100111000000100000000100000000001111111111100111
11100111000000000000000010000000001111111011100101
11100111000000000000000000010000001111111111100111
11100111000000001000000000000000001111111110100111
11100011000000000000000000000000000000000011100111
01100101000000000010000000000000000000000011100111
11100111000000000000010000000000000000000011100111
10100111000000000000000000000000000000000011100111
11100111000000000000000000000000000000001011100110
01100111000001000000010000000000000000100011100111
11100111000000000000000000000001000000100011100111
11100111000000000000000000000000000010000011100111
11100111000000000000000000000001000100000011110111
11100111000000000000000000000100000000000011100111
11100111000000000000000000000000000000000011100111
11100101000000000000000000000000000000010011100111
11100111000000000000000000000100000000000011110111
11100111011000000000000000000000000000000011100111
11100111000001000000000000000000000000000011100111
11100101000000000000000000000000001000000011100111
11100111000000000000000000000000000000000011000111
11100111000000000000000100000100000000001011100111
10100111000000010000000000000000000000000011100111
11100111000100100000000000000000000000000011100111
11100101010001000000000000000000000000000001100111
11100111000000000000000000000000000000000011100111
11100111000000000000000000000000000000000011100111
11100111000000000000000000000000000000000011100111
11100111000000100100000000000000000000000011100111
11100111000000000000010000000000000000000011100111
11100111111111101111111111111111111111111111100111
11100111111110111111111111111111101011111111111100111
11100111111101111111111111111111111011111111111100111
11100000000000000000000000000000000001000000000111
```

```
11100000000000000000000000000010000000000000000111
10111111111111111111111111111111111111111111111111
11111111111111111111111111111111111101111111111111
11111111111111111111111111111111111111111111111111
0
```

Sample Output for the Sample Input

```
3
5
```

Problem F
Houses
Time Limit: 5 seconds

We want to build houses on a rectangular filed. The size of the field is m by n units. Here m denotes the number of columns and n denotes the number of rows. The size of a houses is 1 by 2 units, or 2 by 1 units. Each house occupies two adjacent unit squares and the houses cannot overlap. In addition, there are k unit squares in the field that are not sutiable for building, so you cannot build houses on them. We will call these squares *bad squares*. Given the size of the filed, the locations of bad squares, please compute the maximum number of houses that can be built in the field.

Let the location of the bottom left square be $(0, 0)$. The following figure shows a 3 by 2 field with a bad square at location $(2, 1)$, and we can build at most two houses in this field.

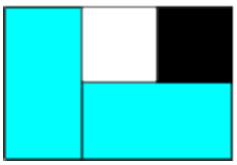


Figure 4: A 3 by 2 field with two houses

Input File

The input file contains a set of test cases, and two test cases are separated by a blank line. Each test case consists of two parts. The first part has one line continaing the size of the field m and n ($1 \leq m, n \leq 200$), and the number of bad squares k. The second part has k lines. Each line has two indices x and y indicating the location of a bad square, where $0 \leq x < m$ and $0 \leq y < n$. We assume that when m, n, or k is 0 then it indicates the end of test cases.

Output Format

The output for each test case is a line that has the maximum number of houses that could be built for this test case.

Sample Input

```
3 2 1
2 1

0 0 0
```

Sample Output for the Sample Input

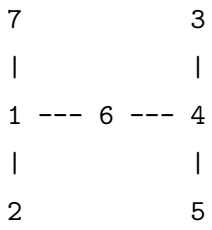
```
2
```

Problem G
Tree Representation
Time Limit: 5 seconds

Given a labeled tree T with n vertices, repeat the following step: *delete the leaf with the smallest label and record the label of its parent until only a single edge remains.* The resulting sequence of $n-2$ labels can be used to represent the tree.

To make life easier, let the labels of the tree be $1, \cdots, n$. Observe that every tree has at least two nodes of degree 1 (i.e., tree leaves). More specifically, the node v which is incident to the lowest labeled leaf is uniquely determined, and v is then taken as the first symbol in the sequence. This lowest labeled leaf is then deleted and the procedure is repeated until a single edge is left, giving a total of $n-2$ integers between 1 and n. Note the important fact that the number of occurrences of the label i in the sequence is one less than the degree of the vertex i in the original tree.

For example, the sequence 1, 4, 4, 6, 1 represents the following labeled spanning tree. Vertex 4 has degree 3 and appears twice in the above sequence.



Now given a sequence of $n-2$ numbers from the set of labels $\{1, 2, \ldots, n\}$, your task is to write a program to recover the tree. Note that any sequence is associated with a unique labeled tree.

Input File

The input file contains a set of test data. The first line of the input file contains a positive integer $m(\leq 5)$ indicating the number of test cases to follow. Each test case starts with a positive integers $n(5 \leq n \leq 50)$ indicating the tree size and then $(n-2)$ integers from $\{1, 2, \cdots, n\}$ follow.

Output Format

For each test case, output the adjacent list of the tree, where the vertices are listed in increasing order. Likewise the neighbor(s) of all vertices are ordered in increasing order. Answers for adjacent cases are separated by a blank line.

Sample Input

```
2
7 1 4 4 6 1
5 1 1 1
```

Sample Output for the Sample Input

```
1  2 6 7
2  1
3  4
4  3 5 6
5  4
6  1 4
7  1

1  2 3 4 5
2  1
3  1
4  1
5  1
```

Problem H
Q-Bacteria
Time Limit: 3 seconds

A new form of bacteria, named Q, is discovered by the famous scientist Dr. Truth. It is known that Q-bacteria live in a closed environment, called *Q-community*, without any interaction with other Q-communities. In a community, Q-bacteria stay together in groups. Depending on their DNA structure, a Q-bacterium may or may not be able to transfer a chemical component, called *Famma*, to another Q-bacterium. It is also known that Q-bacterium Q_1 can transfer Famma to Q-bacterium Q_2 if and only if Q_2 can also transfer Famma to Q_1. The chemical component Famma can be synthesized in units of cells through a complicated process, called *Malus*, that is invented by Dr. Truth. When a cell of Famma is first synthesized, it is *fresh*. A Q-bacterium can process any cell of fresh Famma. A cell of Famma can be possessed by at most one Q-bacterium at any time. There is no limit on the number of cells of Famma that a Q-bacterium can possess. After a fresh cell of Famma is possessed by a Q-bacterium, it is called *used*. Used Famma can only be transferred to another Q-bacterium. A cell of used Famma in a Q-bacterium does not (1) disappear, (2) split into parts, (3) combine with other Famma cell, or (4) drop out of the body of a Q-bacterium unless it is being transferred to another Q-bacterium.

When a Q-bacterium carries a cell of used Famma, it will try to transfer or send away this cell of Famma to another bacterium because this is the way a Q-bacterium to show friendship to fellow Q-bacteria. However, if a cell of used Famma W is currently being carried by a Q-bacterium Q_1 and is transferred to another Q-bacterium Q_2, then Q_1 does not take back W if it is being transferred from Q_2. However, Q_1 can carried W again if it is being transferred from a Q-bacterium that is not Q_2.

Q-bacteria are bad to the health of human and cannot be killed by any known antibiotic. Fortunately, Q-bacteria have built into their DNA structures a property as discovered by Dr. Truth. If a Famma cell W is being transmitted from Q-bacterium Q_1 to Q_2, Q_2 will die if Q_2 has carried W before and W has been carried by an odd, but greater than 1, number of different Q-bacteria, including Q_2, between now and the previous time Q_2 carried it. When a Q-bacterium is dead, every Q-bacterium in the community will die because the body of a dead Q-bacterium D caused by Famma releases a chemical component called *Verum-i* that is lethal to all Q-bacteria within the same community D lives. Up to today, this is the only known way to kill Q-bacteria.

Given the number Q-bacteria in a community, and their DNA characteristics, you task is find whether Q-bacteria in this community can be destroyed by exposing to Famma. If it is possible to destroy all Q-bacteria in a Q-community by dropping Famma, you need to compute the shortest possible time that a Q-bacterium is found dead assuming that it takes one time unit to transfer Famma from one Q-bacterium to another Q-bacterium and it takes no time at all to kill a Q-bacterium because of the miracle mathematical taste.

Input File

The input file contains a set of test data. Each test data consists of two parts representing a Q-community. The first part is the number of Q-bacteria n. Assume that $0 \leq n \leq 100$, and $n = 0$ signals the end of the test data.

The second part is a sequence of n lines. The ith line contains the DNA characteristics for the ith bacterium in the form of $a_{i,1}\ a_{i,2}\ \ldots\ a_{i,n}$ where $a_{i,j}$ is 1 if the ith bacterium can transfer Famma to the jth bacterium; it is 0 if otherwised; Note that there is a single blank between $a_{i,j}$ and $a_{i,j+1}$ for each $1 \leq j < n$.

Output Format

The output file contains exactly n lines. The ith line contains the shortest possible time that a Q-bacterium is found dead after dropping Famma. If the ith input Q-community cannot be destroyed, please output 0.

Sample Input

```
3
0 1 1
1 0 1
1 1 0
4
0 1 0 1
1 0 1 0
0 1 0 1
1 0 1 0
0
```

Sample Output for the Sample Input

```
3
0
```

Problem I
History of Dots
Time Limit: 2 seconds

Dots are peculiar two dimensional creatures. Since the creation of the dot universe, each dot resides in a unique and isolated community, all interactions and communications between dots are limited to the dots in the same community; also, the number of dots never changes, i.e., no birth nor death will ever occur for dots. Dots come in assorted colors, when two dots of different colors meet they would transform to another pair of colors according to the transformation rules in the book of life. More often than not, a dot community would reach the eternal state and never transform again, such as, when all dots in a community are settled in one identical color, then no more transformations could ever happen again. It is still a mystery how the dots come into being (rumor has it that they are created during an ACM programming contest). As the dots stay in the eternal state, they have endless time to speculate their origin. Since it is very difficult to study the moment of creation, they decide to first study the probable initial population states, that is the number of dots in each color at the beginning. Even this is not an easy task, first of all they have to find out the color transformation rules, and secondly they have to find out as much as possible about the past so that they can rule out any impossible initial population. Luckily, they do manage to find out certain historical facts of their communities in the form of a sequence of transient population states, so that they know that some specific population states appeared in a specific order in time. Your mission is to write a program, given the number of dots, the number of colors, the color in the eternal state, the color transformation rules, a set of initial populations, and perhaps a series of past transient population states, to determine which initial population is possible and which is not.

Let n ($1 \leq n \leq 30$) be the number of dots, k ($1 \leq k \leq 5$) be the number of colors, and $C = \{1, 2, ..., k\}$ be the set of colors. Each color transformation rule is presented in the form of $c_i\ c_j\ c_{i'}\ c_{j'}$, which means that when a dot in color c_i and a dot in color c_j meet ($i \neq j$), they transform to dots in color $c_{i'}$ and $c_{j'}$. There is no transformation when two dots of the same color meet. Only rules result in a transformation are listed in the book of life. The color for the final eternal state will be $c_e \in C$. The initial population and each of the transient population states (at most 5) are presented by $(n_1, n_2, ..., n_k)$ where n_j ($1 \leq j \leq k$) is the number of dots in color j and $\sum_{j=1}^{k} n_j = n$.

Input Format

The input file is a text file with two test cases. Each test case comprises two text lines, with the first line listing the number of dots (n), the number of colors (k), the color of dots in the eternal state (c_e), a series of transient population states (at most 5), and the transformation rules. The second line is a series of possible initial population states (at most 10 per test case).

Output Format

For each test case, output a sequence of 0 and 1 in one line, where 0 indicates that corresponding initial population is not possible, and 1 means possible.

Sample Input

```
4 2 2 (2,2) 1 2 2 2
```

```
(4,0)(2,2)(0,4)
6 3 3 (2,3,1) 1 2 2 2 2 3 3 3
(4,2,0)(3,2,1)(2,2,2)(1,4,1)
```

Sample Output for the Sample Input

```
0 1 0
0 1 0 0
```

Problem J
Falling Gift Game
Time Limit: 5 seconds

There are falling gifts from a game machine. Each gift is labeled with a price (see Fig. 1). The gifts fall at different timing which people do not know. The game allows you to move a cart at the floor of the game machine to collect the gifts from left to right using a control button (starting from line 1). When you press the button, the cart will move right. Moving from a line to it next line will take 1 second. You cannot move backward. Of course, you can wait at a line to guarantee a catch. There is no need to move the car to the end of right before the game ends.

In a strange occasion, you got the falling timing setting of the machine. Let the game start at 0 second. By calculating how much time it takes to fall, you exactly know when a gift will hit the floor of the game machine. You are a greedy man and decide to come up a great plan to get the maximum value from the machine.

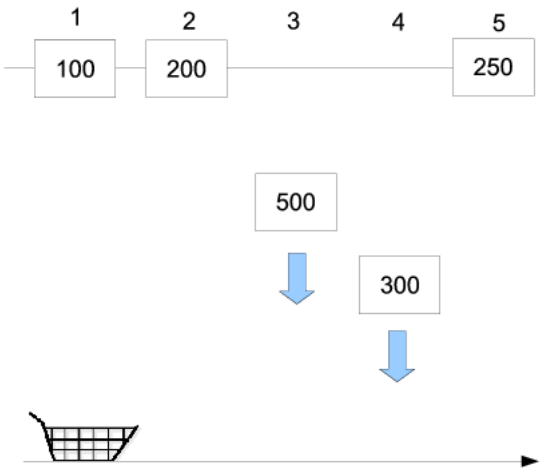


Figure 5: The game machine.

Input File Format

In each test case, the first line is the number of gifts G ($G < 500$). If G is 0 it means the end of input data. Next, there are G lines of gift information. The gift information is listed from left to right. Each gift information is described by time t and price p. The value of t, $1 \leq t \leq 2G$, is the time the gift will hit the floor of the game machine. Every gift will fall to the floor within $2G$ seconds. The value of p is the price of the gift, which is a positive integer, $1 \leq p \leq 10000$. For example, $t = 3$ and $p = 100$ mean the gift will reach the floor at the 3-rd second when the game starts and the price will be 100.

Output Format

Please output the maximum value you can collect in a game.

Sample Input

```
5
5 100
4 200
3 500
4 300
4 250
3
5 500
3 300
4 300
5
2 200
4 200
5 200
3 500
7 50
0
```

Sample Output for the Sample Input

```
800
600
650
```