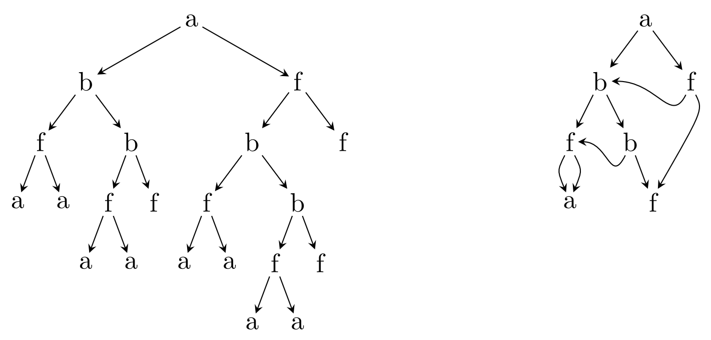
Problem B
Common Subexpression Elimination

Let the set Σ consist of all words composed of 1–4 lower case letters, such as the words "a", "b", "f", "aa", "fun" and "kvqf". Consider expressions according to the grammar with the two rules

$$E \to f$$
$$E \to f(E, E)$$

for every symbol $f \in \Sigma$. Any expression can easily be represented as a tree according to its syntax. For example, the expression "a(b(f(a,a),b(f(a,a),f)),f(b(f(a,a),b(f(a,a),f)),f))" is represented by the tree on the left in the following figure:



Last night you dreamt of a great invention which considerably reduces the size of the representation: use a graph instead of a tree, to share common subexpressions. For example, the expression above can be represented by the graph on the right in the figure. While the tree contains 21 nodes, the graph just contains 7 nodes.
Since the tree on the left in the figure is also a graph, the representation using graphs is not necessarily unique. Given an expression, find a graph representing the expression with as few nodes as possible!

Input

The first line of the input contains the number c ($1 \leq c \leq 200$), the number of expressions. Each of the following c lines contains an expression according to the given syntax, without any whitespace. Its tree representation contains at most 50 000 nodes.

Output

For each expression, print a single line containing a graph representation with as few nodes as possible.
The graph representation is written down as a string by replacing the appropriate subexpressions with numbers. Each number points to the root node of the subexpression which should be inserted at that position. Nodes are numbered sequentially, starting with 1; this numbering includes just the nodes of the graph (not those which have been replaced by numbers). Numbers must point to nodes written down before (no forward pointers). For our example, we obtain "a(b(f(a,4),b(3,f)),f(2,6))".

Sample Input

```
3
this(is(a,tiny),tree)
a(b(f(a,a),b(f(a,a),f)),f(b(f(a,a),b(f(a,a),f)),f))
z(zz(zzzz(zz,z),zzzz(zz,z)),zzzz(zz(zzzz(zz,z),zzzz(zz,z)),z))
```

Sample Output

```
this(is(a,tiny),tree)
a(b(f(a,4),b(3,f)),f(2,6))
z(zz(zzzz(zz,z),3),zzzz(2,5))
```

A

C
Lights

John has n light bulbs and a switchboard with n switches; each bulb can be either on or off, and pressing the $i-$th switch changes the state of bulb i from on to off, and viceversa. He is using them to play a game he has made up. In each move, John selects a (possibly empty) set of switches and presses them, thus inverting the states of the corresponding bulbs. After exactly m moves, John would like to have the first v bulbs on and the rest off; otherwise he loses the game. There is only one restriction: he is not allowed to press the same *set* of switches in two different moves.

This is quite an easy game, as there are lots of ways of winning. This has encouraged him to keep playing different winning games, and now he is intent on trying them all. Help him count how many ways of winning there are. Two games are considered the same if, after a reordering of the moves in one of them, at every step the same set of switches is pressed in both of them.

For example, if $n = 4$, $m = 3$, and $v = 2$, one possible winning game is obtained by pressing switches 1, 2 and 4 in the first move, 1 and 3 in the second one, and 1, 3 and 4 in the last one. This is considered equivalent to, say, first pressing 1 and 3; then 1, 2, 4; and then 1, 3, 4.

Input

The input has at most 500 lines, one for each test case. Each line contains three integers n ($1 \leq n \leq 1\,000$), m ($1 \leq m \leq 1\,000$), and v ($0 \leq v \leq n$). The last line of input will hold the values `0 0 0` and must not be processed.

Output

Print one line for each test case containing the number of ways John can play the game, modulo the prime 10 567 201.

Sample Input

```
3 3 1
6 4 0
6 4 3
0 0 0
```

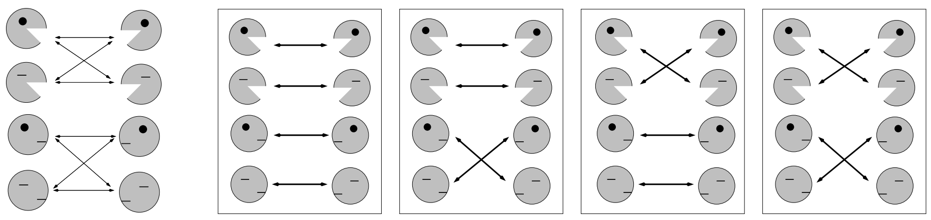
Sample Output

```
7
10416
9920
```

B

F – (False) faces

The Company is testing its brand new face recognition solution. The application is supposed to recognize people given their profiles. It is fed with a number of test cases to check if everything is working fine. Each test consists of $2n$ photos of n persons; there is a left profile and a right profile of each person in a single test case. The program matches left profiles with right profiles, but it is still far from perfect, so sometimes several right profiles are assigned to a single left profile (and vice versa). A *consistent reconstruction* is an assignment of *different* right profiles to *all* left profiles, such that all the pairs matched were proposed by the program.


The program output (on the left) and four possible consistent reconstructions.

In order to test the program, it is necessary to verify every possible consistent reconstruction. The verification has to be done by humans and the Company has a team of four experts who devoted their lives to face recognition. They are willing to do the job under one condition: their shares of work have to be equal, i.e., the number of consistent reconstructions has to be divisible by four. Your task is to check if it is so.

Multiple Test Cases

The input contains several test cases. The first line of the input contains a positive integer $Z \leq 100$, denoting the number of test cases. Then Z test cases follow, each conforming to the format described in section *Single Instance Input*. For each test case, your program has to write an output conforming to the format described in section *Single Instance Output*.

Single Instance Input

The first line contains the number n of persons in a test ($1 \leq n \leq 300$). Then n lines follows, each containing n characters, each of them being 0 or 1. The j-th character in the i-th line is 1 if and only if the program matches the i-th left profile with the j-th right profile.

Single Instance Output

The output should consist of one line containing `YES` if the number of reconstructions consistent with the program assignment is divisible by 4 and `NO` otherwise.

Example

Input	Output
2 4 1100 1100 0011 0011 3 111 011 001	YES NO

C

Problem E
Mountain Road

D

In the Franconian Switzerland, there is a narrow mountain road. With only a single lane, this is a bottleneck for two-way traffic. Your job is to schedule incoming cars at both ends so that the last car leaves the road as early as possible.

Each car is specified by three values: the direction in which it is going, the arrival time at the corresponding beginning of the road, and the driving time this car needs to get through, provided it is not slowed down by other cars in front. Cars cannot overtake each other on the mountain road, and reordering cars in the queues at the ends of the road is not allowed.

For safety reasons, two successive cars going in the same direction may not pass any point of the road within less than 10 seconds. This ensures that the second car will not crash into the first car if the latter brakes hard. However, if another car passes in the other direction in between, it will be clear that the road is empty, so in this case, this rule does not apply.

Input

The first line of the input consists of a single integer c ($1 \le c \le 200$), the number of test cases.

Then follow the test cases, each beginning with a single line consisting of an integer n ($1 \le n \le 200$), the number of cars you are to consider in this test case. The remainder of each test case consists of n lines, one line per car, starting with a single upper case letter (“`A`” or “`B`”), giving the direction in which the car is going. Then follow, on the same line, two integers t ($0 \le t \le 100\,000$) and d ($1 \le d \le 100\,000$), giving the arrival time at the beginning of the road and the minimum travel time, respectively, both in seconds.

Within a test case, the cars are given in order of increasing arrival time, and no two cars will arrive at the same time.

Output

For each test case, print a single line consisting of the point in time (in seconds) the last car leaves the road when the cars are scheduled optimally.

Sample Input

```
2
4
A 0 60
B 19 10
B 80 20
A 85 100
4
A 0 100
B 50 100
A 100 1
A 170 100
```

Sample Output

```
200
270
```

D

E
Genetics

A colony of alien bacteria has recently been discovered close to a crater in New Mexico. Dr. Poucher is in charge of the scientific team at the ICPC BioLab committed to the study of the alien DNA structure. We briefly sketch their discoveries here.

Alien DNA molecules have the structure of a circular sequence. Each sequence is composed of nucleotides. There are 26 different types of nucleotides, and each of them can occur in two faces. It is very important to remark that in any given alien DNA molecule, every nucleotide either does not appear at all or appears exactly twice (hence, the length of a DNA molecule is an even integer between 2 and 52). In case a nucleotide occurs twice, each occurrence can be of either type independently. Alien bacteria have two types of extremities, which in the technical biological jargon are referred to as arms and legs. A major discovery of Dr. Poucher's team is a method to determine the exact number of arms and legs of a bacterium by examining its DNA structure.

Here we represent each nucleotide as a letter of the alphabet. We refer to the different nucleotides as `a`, `A`, ... `z`, `Z`, where the lowercase and uppercase forms of a letter represent the two possible faces a nucleotide may appear with; we shall also use a/A, b/B, ... z/Z to refer to a nucleotide in either face.

To determine the number of extremities, Dr. Poucher starts by initializing two counters of arms and legs to zero, and then proceeds to perform a number of surgeries, transforming a DNA sequence into another one. After each transformation, you may need to increase some of the counters, depending on the type of surgery applied. When the empty sequence of nucleotides (which will be denoted by $\emptyset$) has been reached, the number of extremities of the original molecule has been found. The possible surgeries are:

1. Eliminate consecutive instances of a given nucleotide appearing with opposite faces. The number of arms and legs is preserved. For example: `aBbCaC` $\rightarrow$ `aCaC` by eliminating `Bb`. Another example: `DeHhEd` $\rightarrow$ `eHhE` by eliminating `dD`. Remember that DNA structure is circular, so in our representation as a string the last and first letters are connected.

2. Eliminate consecutive nucleotides appearing with the same face. Add one to the number of arms. For example: `BBcgCg` $\rightarrow$ `cgCg` by eliminating `BB`. Another example: `xabyyaBX` $\rightarrow$ `xabaBX` by eliminating `yy`.

3. Eliminate a sequence of four nucleotides formed by two different nucleotides that appear alternately where different occurrences of the same nucleotide have opposite faces. Add one to the number of legs. For example: `dcDCefFe` $\rightarrow$ `efFe`, by eliminating `dcDC`. Another example: `cmNMnC` $\rightarrow$ `cC` by eliminating `mNMn`.

4. Cut and paste, the most sophisticated procedure. First, a nucleotide is selected, for instance a/A, and the DNA sequence is chopped into two linear chains such that the nucleotide appears once in each of them.

   Second, if both occurrences of a/A are of the same face, one of the chains is "inverted" by reversing the sequence and changing the face of every nucleotide in the chain.

   Then, the chains are combined by concatenating the subsequence occurring before `a` with the sub-sequence occurring after `A`, and the subsequence occurring after `a` with the sub-sequence occurring before `A`.

   Finally, two new a/A nucleotides are added to close the chain into a circular shape. The face of the new nucleotides are the same if the original pair of nucleotides selected had the same face, and is different otherwise.

   Formally, suppose you select the nucleotide a/A, and further assume for the moment that it appears both times with the face `a` (`A`). The cut and paste surgery turns sequences of the form S_1`a`S_2S_3`a`S_4

(respectively $S_1\mathtt{A}S_2S_3\mathtt{A}S_4$) into $S_2\mathtt{a}S_1\bar{S}_3\mathtt{a}\bar{S}_4$ (respectively $S_2\mathtt{A}S_1\bar{S}_3\mathtt{A}\bar{S}_4$). On the other hand, if nucleotide a/A appears with its two different faces, the surgery turns sequences of the form $S_1\mathtt{a}S_2S_3\mathtt{A}S_4$ into $S_2\mathtt{a}S_1S_4\mathtt{A}S_3$. S_1, S_2, S_3 and S_4 are arbitrary sub-chains (possibly empty). In both cases the original circular chain was chopped into $S_1(a/A)S_2$ and $S_3(a/A)S_4$.

For example (see the figure below): starting with the sequence `BacDcAbD`, we can get chains `BacDc` and `AbD`. Then, merging at nucleotide a/A we get the sequence `cDca'BbDA'` where `a'` and `A'` represent the new a/A nucleotides. Here, $S_1 = \mathtt{B}$, $S_2 = \mathtt{cDc}$, $S_3 = \emptyset$ and $S_4 = \mathtt{bD}$.

Another example: take the same DNA sequence `BacDcAbD`, and cut to get the chains `DBac` and `DcAb`; paste nucleotide c/C (in this case you need to reverse one chain, for example `BaCd`) to get the sequence `cDBadcBa`. Here, $S_1 = \mathtt{DBa}$, $S_2 = \emptyset$, $S_3 = \mathtt{D}$ and $S_4 = \mathtt{Ab}$.

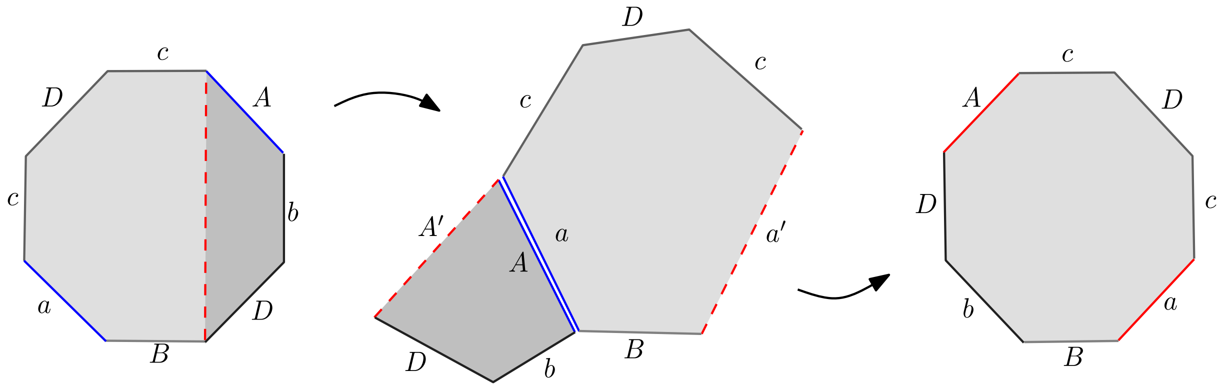


Figure 2: First example of use of cut&paste

This surgery does not modify the number of arms or legs, but can be used cleverly in combination with the previous surgeries to reduce the size of the DNA molecule and finish the calculation.

However, alien bacteria do not present both arms and legs at the same time. This is due to the fact that, in their early development, a leg, in the presence of one or more arms, becomes two arms. Because of the above, the end result is either a number of arms or a number of legs, but not both at the same time. In order to avoid expensive surgical procedures, Dr. Poucher has hired you to write a program that computes the number of arms and legs a bacterium will develop, given its DNA sequence. It is guaranteed that the result is determined uniquely by the original string, regardless of the particular sequence of surgeries applied.

Input

Each test case consists of a string of even length between 2 and 52, inclusive, representing the DNA structure of an alien bacterium. All characters are letters. There will be one case per line in the input. The last line contains the word "END" and must not be processed.

Output

The output for each test case should have exactly one line, containing the number of arms or legs the bacterium will have, followed by the word "arms" or "legs" respectively (if the number is 1, the words should be in singular). In case there will be neither arms nor legs, the program should print the word "none".

Sample Input

```
rkrk
abcdeABCDE
shcoOCfFHS
END
```

Sample Output

```
1 arm
2 legs
none
```

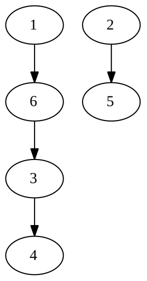
E

H. Hypervisor MacrOS

Bob is the leader of a team developing a hypervisor system *MacrOS*. The project is huge and comprises lots of packages. Since some of them may depend on others, the installation process of the whole system is rather complicated. MacrOS is in alpha stage now, so there are many customers testing the new product. The installation process is overwhelming for many of them, and unfortunately the technical support is Bob's responsibility as well.

Bob had been given a list of dependencies by his team of programmers before the alpha tests started. Each dependency reads "package B depends on A", i.e., one has to install package A before B. This is denoted `A B` in short. Of course, if package C depends on B, and B in turn depends on A, then C depends on A as well, i.e., the dependencies are transitive. For example, the initial list of dependencies can be as follows:

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



As the product develops, the programmers sometimes call Bob to inform of new dependencies. Further, being in charge of technical support, Bob often receives phone calls from customers with questions which packages should be installed first. To no surprise, after several calls from programmers and customers, Bob realized how difficult it is to keep track of dependencies and answer queries on-line at the same time, and. For automatizing this process, he wrote a program which generated two log files. The first one contains the history of all phone calls. An entry `1 A B` denotes a new dependency introduced by the programmers, and `0 A B` denotes a query from a customer meaning "should I install A before B?". The second log is a history of all answers given to customers. An example is given in Table 1.

After a long period of testing, MacrOS is finally ready to enter beta stage. But Bob wants to be sure that there were no mistakes during the alpha testing. He wants to check if the answers given in the second log file are correct, and you are to help him. However, Bob does not give you the second log file. Moreover, he modified the first log in a tricky way: after each line corresponding to customer phone call that should have been answered with `NO`, he started/stopped reversing all lines corresponding to programmers calls; see the example below. Taking into account the Bob's modification of the first log file, you should give all the answers to the customers' questions.

First log file	Second log file (answers)
1 1 3	
0 1 3	YES
1 1 4	
0 5 2	NO
1 3 2	
1 6 5	
0 1 5	YES
1 4 5	
0 5 3	NO
1 1 2	

Table 1. The first answer is `YES` because package `1` should be installed before `3`.

Modified first log file
1 1 3
0 1 3
1 1 4
0 5 2 – start reversing
1 **2 3**
1 **5 6**
0 1 5
1 **5 4**
0 5 3 – stop reversing
1 1 2

Table 2. Reversed numbers of packages are written in bold.

Multiple Test Cases

The input contains several test cases. The first line of the input contains a positive integer $Z \leq 15$, denoting the number of test cases. Then Z test cases follow, each conforming to the format described in section *Single Instance Input*. For each test case, your program has to write an output conforming to the format described in section *Single Instance Output*.

Single Instance Input

Two integers n and m ($1 \leq n \leq 10^5$, $0 \leq m \leq 10^5$) separated by a single space are given in the first line of an input instance. These denote the number of MacrOS' packages and the number of dependencies before alpha tests took place, respectively. The following m lines describe the initial list of dependencies. Each contains two numbers A and B ($1 \leq A, B \leq n$) separated by a single space, denoting that package B depends on A. Some dependencies may appear multiple times, but there will be no pair of mutually dependent packages. The next part of the input contains the Bob's modified version of the first log file, with the format described above. The total number of all phone calls is at most 10^5. The input instance ends with a line containing three zeros, `0 0 0`.

Single Instance Output

Your program should print the content of the second log file. This means that for each input line `0 A B`, you should print `YES` if package A should be installed first and `NO` if B should be installed first. **You can assume that all the queries have a unique answer, i.e., at the moment of such query there was already a dependency between packages A and B.**

Example

Input

```
2
6 4
1 6
6 3
3 4
2 5
1 1 3
0 1 3
1 1 4
0 5 2
1 2 3
1 5 6
0 1 5
1 5 4
0 5 3
1 1 2
0 0 0
6 3
1 2
5 3
4 6
1 2 5
0 1 5
1 4 1
0 3 4
0 0 0
```

Output

```
YES
NO
YES
NO
YES
NO
```

Problem G
Room Assignments

Once there was an inventor congress, where inventors from all over the world met in one place. The organizer of the congress reserved exactly one hotel room for each inventor. Each inventor, however, had its own preference regarding which room he would like to stay in. Being a clever inventor himself, the organizer soon found an objective way of doing the room assignments in a fair manner: each inventor wrote two different room numbers on a fair coin, one room number on each side. Then, each inventor threw his coin and was assigned the room number which was shown on the upper side of his coin. If some room had been assigned to more than one inventor, all inventors had to throw their coins again.
As you can imagine, this assignment process could take a long time or even not terminate at all. It has the advantage, however, that among all possible room assignments, one assignment is chosen randomly according to a uniform distribution. In order to apply this method in modern days, you should write a program which helps the organizer.
The organizer himself needs a hotel room too. As the organizer, he wants to have some advantage: he should be able to rate each of the rooms (the higher the rating, the better), and the program should tell him which two room numbers he should write on his coin in order to maximize the expected rating of the room he will be assigned to. The program also has access to the choices of the other inventors before making the proposal. It should never propose two rooms for the organizer such that it is not possible to assign all inventors to the rooms, if a valid assignment is possible at all.

Input

The input starts with a single number c ($1 \leq c \leq 200$) on one line, the number of test cases. Each test case starts with one line containing a number n ($2 \leq n \leq 50\,000$), the number of inventors and rooms. The following $n-1$ lines contain the choices of the $n-1$ guests (excluding the organizer). For each inventor, there is a line containing two numbers a and b ($1 \leq a < b \leq n$), the two room numbers which are selected by the inventor. The last line of each test case consists of n integers $v_1, \ldots, v_n$ ($1 \leq v_i \leq 1\,000\,000$), where v_i is the organizer's rating for room i.

Output

For each test case, print a single line containing the two different room numbers a and b which should be selected by the organizer in order to maximize the expected rating of the room he will be assigned to. If there is more than one optimal selection, break ties by choosing the smallest a and, for equal a, the smallest b. If there is no way for the organizer to select two rooms such that an assignment of inventors to rooms is possible, print "`impossible`" instead.

Sample Input

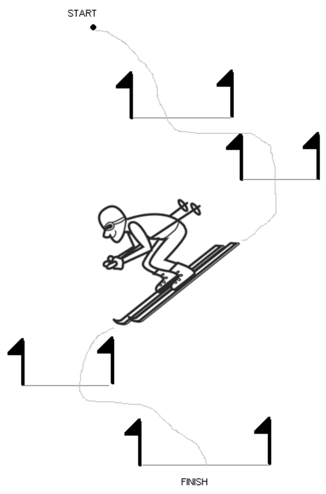
```
3
4
1 2
2 3
1 3
2 3 4 1
3
1 2
2 3
100 40 70
5
1 2
1 2
1 2
3 4
1 1 1 1 1
```

Sample Output

```
1 4
1 3
impossible
```

G

G
Slalom



In spite of the scarcity of snowfall in Madrid, interest in winter sports is growing in the city, especially with regard to skiing. Many people spend several weekends or even full weeks improving their skills in the mountains.

In this problem we deal with only one of the multiple alpine skiing disciplines: slalom. A course is constructed by laying out a series of gates, which are formed by two poles. The skier must pass between the two poles forming each gate. The winner is the skier who takes the least time to complete the course while not missing any of the gates.

You have recently started to learn to ski, but you have already set yourself the goal of taking part in the Winter Olympic Games of 2018, for which Madrid will presumably present a candidature. As part of the theoretical training, you need to write a program that calculates, given a starting point and a series of gates, the minimum-length path starting from the point given and passing through each gate until you reach the last one, which is the finish line. You may assume that the gates are horizontal and are ordered from highest to lowest, so that you need to pass through them in order. You consider yourself an accomplished skier, so you can make any series of turns, no matter how difficult, and your only concern is minimizing the total length of the path.

Input

The first line of each case gives the number of gates n ($1 \leq n \leq 1\,000$). The next line contains two floating point numbers, the Cartesian coordinates x and y of the starting position, in that order. Next come n lines with three floating point numbers each, y x_1 x_2, meaning that the next gate is a horizontal line from (x_1, y) to (x_2, y). You can safely assume that $x_1 < x_2$. The values of y are strictly decreasing and are always smaller than that of the starting position. The last gate represents the finish line. All coordinates are between $-500\,000$ and $500\,000$, inclusive. A value of 0 for n means the end of the input. A blank line follows each case.

Output

For each test case, output a line with the minimum distance needed to reach the finish line. Your answer should be accurate to within an absolute or relative error of 10^{-7}.

Sample Input

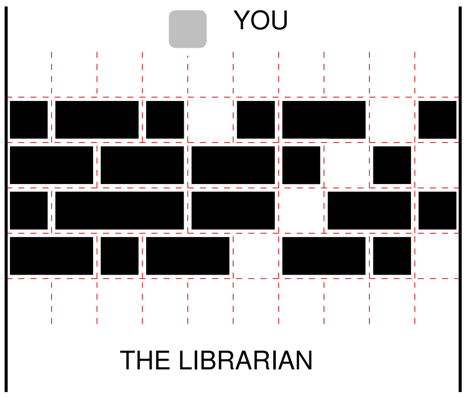
```
2
0 2
1 1 2
0 0.5 3

3
0 4
3 1 2
2 -1 0
1 1 2

0
```

Sample Output

```
2.41421356237
4.24264068712
```

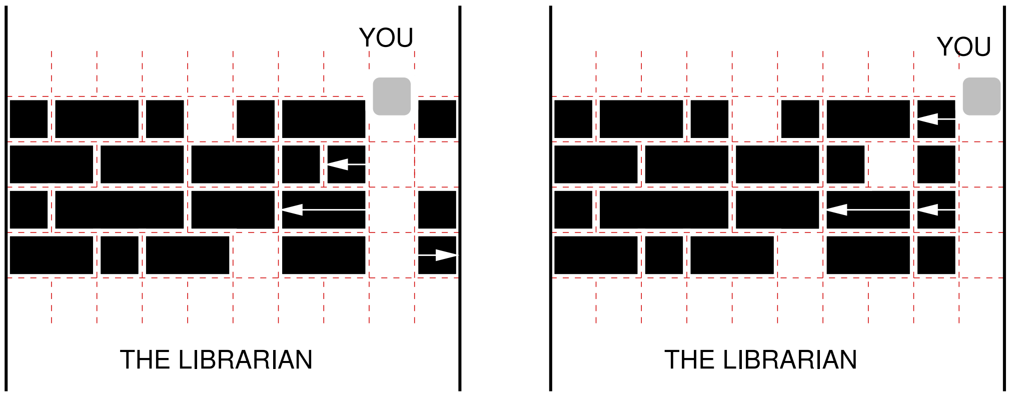
K - Knowledge for the masses

You are in a library equipped with bookracks that move on rails. There are many parallel rails, i.e., the bookracks are organized in several rows, see figure:



The boockracks in the library. There is no passage to the librarian at the moment.

To borrow a book, you have to find the librarian, who seems to hide on the opposite side of the bookracks. Your task then is to move the racks along the rails so that a passage forms. Each rack has a certain integer width, and can be safely positioned at any integer point along the rail. (A rack does not block in a non-integer position and could accidentally move in either direction). The racks in a single row need not be contiguous — there can be arbitrary (though integer) space between two successive bookracks. A passage is formed at position k if there is no bookrack in the interval $(k, k+1)$ in any row (somehow you don't like the idea of trying to find a more sophisticated passage in this maze.)



The passages formed in the library: at position 8 (the left figure) and at position 9 (the right figure). Both attained at cost 3 by moving the bookracks marked with arrows.

Moving a rack requires a certain amount of effort on your part: moving it in either direction costs 1. This cost does not depend on the distance of the shift, which can be explained by a well known fact that static friction is considerably higher than kinetic friction. Still, you are here to borrow a book, not to work out, so you would like to form a passage (at any position) with as little effort as possible.

Multiple Test Cases

The input contains several test cases. The first line of the input contains a positive integer $Z \leq 15$, denoting the number of test cases. Then Z test cases follow, each conforming to the format described in section *Single Instance Input*. For each test case, your program has to write an output conforming to the format described in section *Single Instance Output*.

Single Instance Input

Two space separated integers R and L ($1 \leq R$, $1 \leq L \leq 10^6$) are given in the first line of an input instance. They denote the number of rows and the width of each and every row, respectively. Then R lines with rows descriptions follow. Each such line starts with an integer n_i, followed by n_i integers $a_{i,1}, a_{i,2}, \ldots a_{i,n_i}$, all separated by single spaces. Number $a_{i,j}$ denotes either the width of a bookrack when $a_{i,j} > 0$ or a unit of empty space when $a_{i,j} = 0$. Note that for any row i, $\sum_j a_{i,j}$ equals L minus the number of $a_{i,j}$ that are equal to zero. You may assume that $n_1 + n_2 + \cdots + n_R \leq 2 * 10^7$. Moreover, there will be at least one 0 in the description of each row, which means that creating a passage is always possible.

Single Instance Output

In the first line, your program should output the minimum cost of making a passage through the bookracks. In the second line, it should print out the increasing sequence of all the positions at which a minimum cost passage can be formed.

Example

Input	Output

1
4 10
8 1 2 1 0 1 2 0 1
7 2 2 2 1 0 1 0
6 1 3 2 0 2 1
7 2 1 2 0 2 1 0
	3
	8 9

Problem J
Wormholes

A friend of yours, an inventor, has built a spaceship recently and wants to explore space with it. During his first voyages, he discovered that the universe is full of wormholes created by some alien race. These wormholes allow one to travel to places far, far away, but moreover, they can also send you to times long ago or in the distant future.
Having mapped these wormholes and their respective end points, you and your friend boldly decide to board his spaceship and go to some distant place you'd like to visit. Of course, you want to arrive at your destination as early as possible. The question is: what is this earliest arrival time?

Input

The first line of input contains an integer c ($1 \leq c \leq 200$), the number of test cases. Each test case starts with a line containing two coordinate triples x_0, y_0, z_0 and x_1, y_1, z_1, the space coordinates of your departure point and destination. The next line contains an integer n ($0 \leq n \leq 50$), the number of wormholes. Then follow n lines, one for each wormhole, with two coordinate triples x_s, y_s, z_s and x_e, y_e, z_e, the space coordinates of the wormhole entry and exit points, respectively, followed by two integers t, d ($-1\,000\,000 \leq t, d \leq 1\,000\,000$), the creation time t of the wormhole and the time shift d when traveling through the wormhole.
All coordinates are integers with absolute values smaller than or equal to 10 000 and no two points are the same.
Note that, initially, the time is zero, and that tunneling through a wormhole happens instantly. For simplicity, the distance between two points is defined as their Euclidean distance (the square root of the sum of the squares of coordinate differences) rounded up to the nearest integer. Your friend's spaceship travels at speed 1.

Output

For each test case, print a single line containing an integer: the earliest time you can arrive at your destination.

Sample Input

```
2
0 0 0 100 0 0
2
1 1 0 1 2 0 -100 -2
0 1 0 100 1 0 -150 10
0 0 0 10 0 0
1
5 0 0 -5 0 0 0 0
```

Sample Output

```
-89
10
```

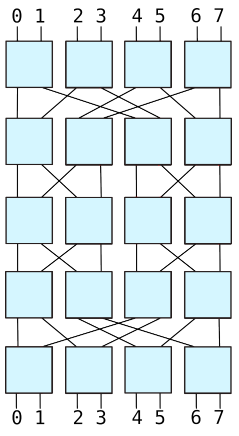
This page is intentionally left (almost) blank.

H Routing

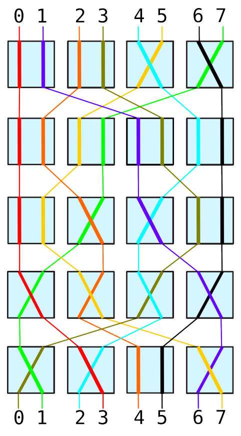
You work as an engineer for the *Inane Collaboration for Performance Computing*, where you are in charge of designing an intercommunication network for their computers. The network is arranged as a rectangular array of $2n-1$ rows, each having 2^{n-1} switches. A switch is a device with two input wires, X and Y, and two output wires, X' and Y'. If the switch is off, data from input X will be relayed to output X', and data from Y to Y'. If it is on, X will be connected to Y' and Y to X'. Additionally, there are 2^n computers in the topmost and bottommost rows, and messages need to be sent between pairs of them. Notice that data from two different sources cannot share a wire but, of course, both pieces of data can be routed through the same switch on different inputs.

You have come to the conclusion that the network that best suits your purposes has the Beneš topology. A 1-Beneš network is just a switch. For $n > 1$, a n-Beneš network can be constructed recursively as follows:

- In the first (top) row there are 2^{n-1} switches such that switch j ($0 \leq j < 2^{n-1}$) has data inputs from computers $2j$ and $2j+1$ (we label the computers in the topmost and bottommost rows with integers between 0 and $2^n - 1$, inclusive, from left to right).
- Then a *perfect shuffle* permutation is applied to the output wires between the first and the second rows of switches, meaning that output number j in a row is connected to input number j' in the next row, where j' is obtained by rotating the n-bit pattern representing j in binary one bit to the right (again, inputs and outputs are numbered from left to right).
- If $n > 2$, the next rows of switches, up to (and including) the last-but-one, form two $(n-1)$-Beneš subnetworks, one on the left side and the other on the right side.
- Finally, the *inverse* shuffle permutation is applied to the outputs and a last row of switches is added.



(a) Topology



(b) Switch states for second sample input

Figure 4: 3-Benes network

For example, Figure **??** shows the Beneš network for $n = 3$ (squares represent switches; computers in the top and bottom rows are not drawn, but assigned with integers from 0 to 7). Figure **??** shows a possible state of the switches; squares where two of the lines cross are switches that have been turned on. You may verify that this state allows us to simultaneously establish communication paths from computers $0, 1, 2, 3, 4, 5, 6, 7$ at the bottom to $3, 7, 4, 0, 2, 6, 1, 5$ at the top, respectively.

You are given a set of pairs (a, b) of computers to connect simultaneously (where a is a computer in the bottom row and b a computer in the top row) by means of wire-disjoint paths, and you are to find how to select the state of all switches so that this can be accomplished.

Input

The first line of each test case is an integer n ($1 \leq n \leq 13$), meaning that you have 2^n pairs of computers to connect, as described above. A line with $n = 0$ marks the end of the input and should not be processed.

Each line with $n > 0$ will be followed by another line containing 2^n integers. The i-th integer ($0 \leq i < 2^n$) will be the computer in the topmost row that the i-th computer in the bottommost row needs to communicate with.

Output

The output for each case should have $2n-1$ lines, each containing a binary string of length 2^{n-1} indicating, for each switch, whether it must be turned on (1) or off (0).

The input given will always have at least one solution. In case of several solutions, return the lexicographically smallest one. That is, the string in the top row must be lexicographically smallest; in case of a tie, the string in the second row must be lexicographically smallest, and so on.

Outputs for different test cases should be separated by a blank line.

Sample Input

```
2
3 2 1 0
3
3 7 4 0 2 6 1 5
0
```

Sample Output

```
00
11
11

0011
0000
0110
1111
1101
```