



Projects

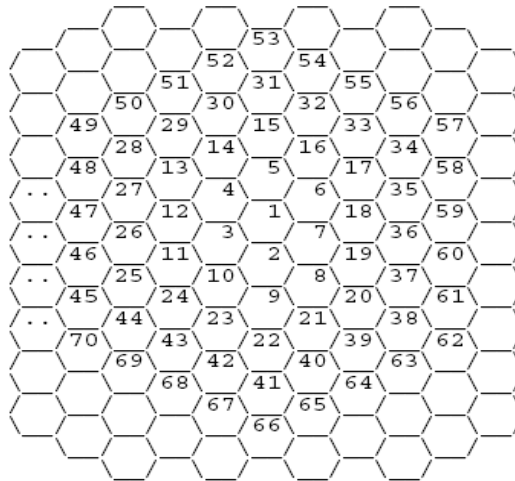
<i>Project No.</i>	<i>Project Name</i>
1	Bee Breeding
2	The Tourist Guide
3	The Grand Dinner
4	Ant on a Chessboard
5	Minesweeper
6	Knight's Tour
7	Injured Queen Problem
8	Balloons in a Box
9	Crossing the Desert
10	Silly Sort
11	Towers of Hanoi
12	Hanoi Tower ²
13	Queue and A
14	Covering Whole Holes
15	Code Generation
16	Watch your move
17	Morse Mismatches
18	Switching Channels
19	Scanner
20	Crossword Puzzle

Problem 1

Bee Breeding

Professor B. Heif is conducting experiments with a species of South American bees that he found during an expedition to the Brazilian rain forest. The honey produced by these bees is of superior quality compared to the honey from European and North American honey bees. Unfortunately, the bees do not breed well in captivity. Professor Heif thinks the reason is that the placement of the different maggots (for workers, queens, etc.) within the honeycomb depends on environmental conditions, which are different in his laboratory and the rain forest.

As a first step to validate his theory, Professor Heif wants to quantify the difference in maggot placement. For this he measures the distance between the cells of the comb into which the maggots are placed. To this end, the professor has labeled the cells by marking an arbitrary cell as number 1, and then labeling the remaining cells in a clockwise fashion, as shown in the following figure.



For example, two maggots in cells 19 and 30 are 5 cells apart. One of the shortest paths connecting the two cells is via the cells 19 - 7 - 6 - 5 - 15 - 30, so you must move five times to adjacent cells to get from 19 to 30.

Professor Heif needs your help to write a program that computes the distance, defined as the number of cells in a shortest path, between any pair of cells.

Input

The input consists of several lines, each containing two integers a and b ($a, b \leq 10000$), denoting numbers of cells. The integers are always positive, except in the last line where $a=b=0$ holds. This last line terminates the input and should not be processed.

Output

For each pair of numbers (a, b) in the input file, output the distance between the cells labeled a and b . The distance is the minimum number of moves to get from a to b .

Sample Input

```
19 30
0 0
```

Output for the Sample Input

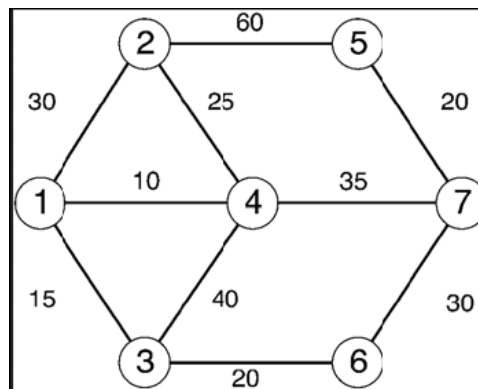
```
The distance between cells 19 and 30 is 5.
```

Problem 2

The Tourist Guide

Mr. G. works as a tourist guide in Bangladesh. His current assignment is to show a group of tourists a distant city. As in all countries, certain pairs of cities are connected by two-way roads. Each pair of neighboring cities has a bus service that runs only between those two cities and uses the road that directly connects them. Each bus service has a particular limit on the maximum number of passengers it can carry. Mr. G. has a map showing the cities and the roads connecting them, as well as the service limit for each bus service.

It is not always possible for him to take all the tourists to the destination city in a single trip. For example, consider the following road map of seven cities, where the edges represent roads and the number written on each edge indicates the passenger limit of the associated bus service.



It will take at least five trips for Mr. G. to take 99 tourists from city 1 to city 7, since he has to ride the bus with each group. The best route to take is 1 - 2 - 4 - 7.

Help Mr. G. find the route to take all his tourists to the destination city in the minimum number of trips.

Input

The input will contain one or more test cases. The first line of each test case will contain two integers: N ($N \leq 100$) and R , representing the number of cities and the number of road segments, respectively. Each of the next R lines will contain three integers ($C1$, $C2$, and P) where $C1$ and $C2$ are the city numbers and P ($P > 1$) is the maximum number of passengers that can be carried by the bus service between the two cities. City numbers are positive integers ranging from 1 to N . The $(R+1)$ th line will contain three integers (S , D , and T) representing, respectively, the starting city, the destination city, and the number of tourists to be guided. The input will end with two zeros for N and R .

Output

For each test case in the input, first output the scenario number and then the minimum number of trips required for this case on a separate line. Print a blank line after the output for each test case.

Sample Input

7 10
1 2 30
1 3 15
1 4 10
2 4 25
2 5 60
3 4 40
3 6 20
4 7 35
5 7 20
6 7 30
1 7 99
0 0

Sample Output

Scenario #1

Minimum Number of Trips = 5

Problem 3

The Grand Dinner

Each team participating in this year's ACM World Finals is expected to attend the grand banquet arranged for after the award ceremony. To maximize the amount of interaction between members of different teams, no two members of the same team will be allowed to sit at the same table. Given the number of members on each team (including contestants, coaches, reserves, and guests) and the seating capacity of each table, determine whether it is possible for the teams to sit as described. If such an arrangement is possible, output one such seating assignment. If there are multiple possible arrangements, any one is acceptable.

Input

The input file may contain multiple test cases. The first line of each test case contains two integers, $1 \leq M \leq 70$ and $1 \leq N \leq 50$, denoting the number of teams and tables, respectively. The second line of each test case contains M integers, where the i th integer m_i indicates the number of members of team i . There are at most 100 members of any team. The third line contains N integers, where the j th integer n_j , $2 \leq n_j \leq 100$, indicates the seating capacity of table j . A test case containing two zeros for M and N terminates the input.

Output

For each test case, print a line containing either 1 or 0, denoting whether there exists a valid seating arrangement of the team members. In case of a successful arrangement, print M additional lines where the i th line contains a table number (from 1 to N) for each of the members of team i .

Sample Input

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

Sample Output

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

Problem 4

Ant on a Chessboard

One day, an ant named Alice came upon an $M \times M$ chessboard. She wanted to explore all the cells of the board. So she began to walk along the board by peeling off a corner of the board.

Alice started at square (1, 1). First, she went up for a step, then a step to the right, and a step downward. After that, she went a step to the right, then two steps upward, and then two grids to the left. In each round, she added one new row and one new column to the corner she had explored. For example, her first 25 steps went like this, where the numbers in each square denote on which step she visited it.

Her 8th step put her on square (2, 3), while her 20th step put her on square (5, 4). Your task is to decide where she was at a given time, assuming the chessboard is large enough to accept all movements.

Input

The input file will contain several lines, each with an integer N denoting the step number where $1 \leq N \leq 2 \times 10^9$. The file will terminate with a line that contains the number 0.

Output

For each input situation, print a line with two numbers (x,y) denoting the column and the row number, respectively. There must be a single space between them.

Sample Input

```
8
20
25
0
```

Sample Output

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

Problem 5

Minesweeper

Have you ever played Minesweeper? This cute little game comes with a certain operating system whose name we can't remember. The goal of the game is to find where all the mines are located within a $M \times N$ field.

The game shows a number in a square which tells you how many mines are there adjacent to that square. Each square has at most eight adjacent squares. The 4×4 field on the left contains two mines, each represented by a "*" character. If we represent the same field by the hint numbers described above, we end up with the field on the right:

* . . .	*100
. . . .	2210
. * . .	1*10
. . . .	1110

Input

The input will consist of an arbitrary number of fields. The first line of each field contains two integers n and m ($0 < n, m \leq 100$) which stand for the number of lines and columns of the field, respectively. Each of the next n lines contains exactly m characters, representing the field.

Safe squares are denoted by "." and mine squares by "*" both without the quotes. The first field line where $n = m = 0$ represents the end of input and should not be processed.

Output

For each field, print the message Field #x: on a line alone, where x stands for the number of the field starting from 1. The next n lines should contain the field with the "." characters replaced by the number of mines adjacent to that square. There must be an empty line between field outputs.

Sample Input

```
4 4
* . . .
. . . .
. * . .
. . . .
3 5
** . . .
. . . . .
. * . . .
0 0
```

Sample Output

```
Field #1:
*100
2210
1*10
1110

Field #2:
**100
33200
1*100
```

Problem 6

Knight's Tour

The Knight's Tour Happens on a (N x N) Chess Board. There are two types of knight's tour, a) Knight's Circuit Tour and b) Knight's Path Tour. Knight's Circuit Tour means that Knight will start from a square, follow Knight's movement rules and jump to another square and thus visit all the squares and come back to its initial square without visiting any square twice. Knight's path tour is very similar to this; the only difference is that knight won't have to come back to its initial square. Below you can see two types of knight's tour in a (6 x 6) board. All the circuit tours are also path tours but not vice versa.

25	32	11	2	19	34
10	1	26	33	12	3
31	24	9	18	35	20
8	17	36	27	4	13
23	30	15	6	21	28
16	7	22	29	14	5

Fig: Knight's Circuit Tour

1	32	9	18	3	34
10	19	2	33	28	17
31	8	29	16	35	4
20	11	36	27	24	15
7	30	13	22	5	26
12	21	6	25	14	23

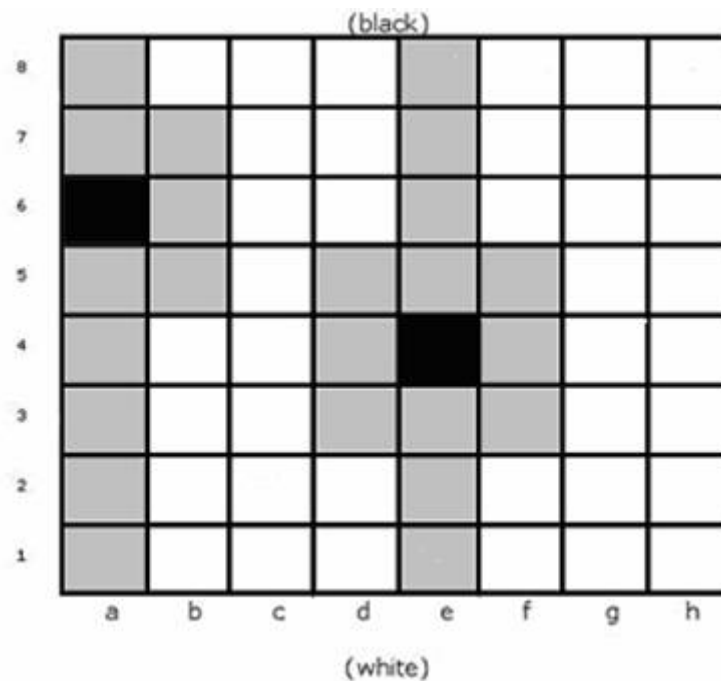
Fig: Knight's Path Tour

Given a board size N you will have to determine whether there exists a Knight's Circuit Tour in that Board. If there is a knight Circuit Tour you will have to print the knight Path Tour in that Board from a given location.

Problem 7

Injured Queen Problem

Chess is a two-player board game believed to have been played in India as early as the sixth century. However, in this problem we will not discuss about chess, rather we will talk about a modified form of the classic n-queen problem. I know you are familiar with plotting n-queens on a chess board with the help of a classic backtracking algorithm. If you write that algorithm now you will find that there are 92 ways of plotting 8 queens in an 8x8 board provided no queens attack each other. In this problem we will talk about injured queens who can move only like a king in horizontal and diagonal direction from current position but can reach any row from current position like a normal chess queen. You will have to find the number of possible arrangements with such injured queens in a particular (n x n) board, such that no two queens attack each other.



Problem 8

Balloons in a Box

You must write a program that simulates placing spherical balloons into a rectangular box.

The simulation scenario is as follows. Imagine that you are given a rectangular box and a set of points. Each point represents a position where you might place a balloon. To place a balloon at a point, center it at the point and inflate the balloon until it touches a side of the box or a previously placed balloon. You may not use a point that is outside the box or inside a previously placed balloon. However, you may use the points in any order you like, and you need not use every point. Your objective is to place balloons in the box in an order that maximizes the total volume occupied by the balloons.

You are required to calculate the volume within the box that is not enclosed by the balloons.

Input

The input consists of several test cases. The first line of each test case contains a single integer n that indicates the number of points in the set ($1 \leq n \leq 6$). The second line contains three integers that represent the (x, y, z) integer coordinates of a corner of the box, and the third line contains the (x, y, z) integer coordinates of the opposite corner of the box. The next n lines of the test case contain three integers each, representing the (x, y, z) coordinates of the points in the set. The box has non-zero length in each dimension and its sides are parallel to the coordinate axes.

The input is terminated by the number zero on a line by itself.

Output

For each test case print one line of output consisting of the test case number followed by the volume of the box not occupied by balloons. Round the volume to the nearest integer. Follow the format in the sample output given below. Place a blank line after the output of each test case.

Sample Input

```
2
0 0 0
10 10 10
3 3 3
7 7 7
0
```

Output for the Sample Input

```
Box 1: 774
```

Problem 9

Crossing the Desert

In this problem, you will compute how much food you need to purchase for a trip across the desert on foot.

At your starting location, you can purchase food at the general store and you can collect an unlimited amount of free water. The desert may contain oases at various locations. At each oasis, you can collect as much water as you like and you can store food for later use, but you cannot purchase any additional food. You can also store food for later use at the starting location. You will be given the coordinates of the starting location, all the oases, and your destination in a two-dimensional coordinate system where the unit distance is one mile.

For each mile that you walk, you must consume one unit of food and one unit of water. Assume that these supplies are consumed continuously, so if you walk for a partial mile you will consume partial units of food and water. You are not able to walk at all unless you have supplies of both food and water. You must consume the supplies while you are walking, not while you are resting at an oasis. Of course, there is a limit to the total amount of food and water that you can carry. This limit is expressed as a carrying capacity in total units. At no time can the sum of the food units and the water units that you are carrying exceed this capacity.

You must decide how much food you need to purchase at the starting location in order to make it to the destination. You need not have any food or water left when you arrive at the destination. Since the general store sells food only in whole units and has only one million food units available, the amount of food you should buy will be an integer greater than zero and less than or equal to one million.

Input

The first line of input in each trial data set contains n ($2 \leq n \leq 20$), which is the total number of significant locations in the desert, followed by an integer that is your total carrying capacity in units of food and water. The next n lines contain pairs of integers that represent the coordinates of the n significant locations. The first significant location is the starting point, where your food supply must be purchased; the last significant location is the destination; and the intervening significant locations (if any) are oases. You need not visit any oasis unless you find it helpful in reaching your destination, and you need not visit the oases in any particular order. The input is terminated by a pair of zeroes.

Output

For each trial, print the trial number followed by an integer that represents the number of units of food needed for your journey. Use the format shown in the example. If you cannot make it to the destination under the given conditions, print the trial number followed by the word "Impossible."

Place a blank line after the output of each test case.

Sample Input

```
4 100
10 -20
-10 5
30 15
15 35
2 100
0 0
100 100
0 0
```

Output for the Sample Input

```
Trial 1: 136 units of food
Trial 2: Impossible
```

Problem 10

Silly Sort

Your younger brother has an assignment and needs some help. His teacher gave him a sequence of numbers to be sorted in ascending order. During the sorting process, the places of two numbers can be interchanged. Each interchange has a cost, which is the sum of the two numbers involved.

You must write a program that determines the minimal cost to sort the sequence of numbers.

Input

The input file contains several test cases. Each test case consists of two lines. The first line contains a single integer n ($n > 1$), representing the number of items to be sorted. The second line contains n different integers (each positive and less than 1000), which are the numbers to be sorted.

The input is terminated by a zero on a line by itself.

Output

For each test case, the output is a single line containing the test case number and the minimal cost of sorting the numbers in the test case.

Place a blank line after the output of each test case

Sample Input	Output for the Sample Input
3 3 2 1	Case 1: 4
4 8 1 2 4	Case 2: 17
5 1 8 9 7 6	Case 3: 41
6 8 4 5 3 2 7	Case 4: 34
0	

Problem 11

Towers of Hanoi

The ancient folklore behind the "Towers of Hanoi" puzzle is quite well known. A more recent legend tells us that once the Brahmin monks discovered how long it would take to finish transferring the 64 discs from the needle which they were on to one of the other needles, they decided to find a faster strategy and be done with it.



One of the priests at the temple informed his colleagues that they could achieve the transfer in single afternoon at a one disc-per-second rhythm by using an additional needle. He proposed the following strategy:

- First move the topmost discs (say the top k discs) to one of the spare needles.
- Then use the standard three needles strategy to move the remaining $n - k$ discs (for a general case with n discs) to their destination.
- Finally, move the top k discs into their final destination using the four needles.

He calculated the value of k which minimized the number of movements and found that 18,433 transfers would suffice. Thus they could spend just 5 hours, 7 minutes, and 13 seconds with this scheme versus over 500, 000 million years without the additional needle!

Try to follow the clever priest's strategy and calculate the number of transfers using four needles, where the priest can move only one disc at a time and must place each disc on a needle such that there is no smaller disc below it. Calculate the k that minimizes the number of transfers under this strategy.

Input

The input file contains several lines of input. Each line contains a single integer $0 \leq N \leq 10,000$ giving the number of disks to be transferred. Input is terminated by end of file.

Output

For each line of input produce one line of output which indicates the number of movements required to transfer the N disks to the final needle.

Sample Input

1
2
28
64

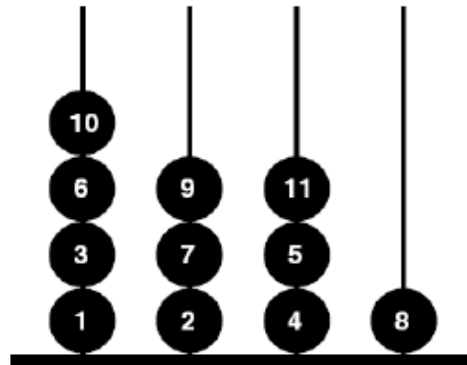
Sample Output

1
3
769
18433

Problem 12

Hanoi Tower²

There are many interesting variations on the Tower of Hanoi problem. This version consists of N pegs and one ball containing each number from $1, 2, 3, \dots, \infty$. Whenever the sum of the numbers on two balls is not a perfect square (i.e., c^2 for some integer c), they will repel each other with such force that they can never touch each other.



The player must place balls on the pegs one by one, in order of increasing ball number (i.e., first ball 1, then ball 2, then ball 3. . .). The game ends where there is no non-repelling move.

The goal is to place as many balls on the pegs as possible. The figure above gives a best possible result for 4 pegs.

Input

The first line of the input contains a single integer T indicating the number of test cases ($1 \leq T \leq 50$). Each test case contains a single integer N ($1 \leq N \leq 50$) indicating the number of pegs available.

Output

For each test case, print a line containing an integer indicating the maximum number of balls that can be placed. Print "-1" if an infinite number of balls can be placed.

Sample Input

```
2
4
25
```

Sample Output

```
11
337
```

Problem 13

Queue and A

The customer support group of Contest.com receives and responds to requests for technical support via e-mail.

Requests may begin arriving when the office opens at 8:00 a.m. and all requests must be serviced by the end of the day.

As requests are received, they are classified according to a predetermined list of topics. Each member of the support staff has responsibility for one or more of these topics and each topic has one or more support personnel assigned to it. Because staff members have different levels of expertise, each staff member has a prioritized list of topics that he or she can handle. Staff personnel are not permitted to handle requests outside their specified areas.

As staff members become available, they select from the pool of waiting requests according to their priority list of topics. All requests arriving at time t are available for allocation at time t . If two staff members are simultaneously available, scheduling preference is given to the one whose most recent job was scheduled earliest. If there is still a tie, scheduling preference is given to the person whose id number appears earlier in the input list of staff people. At the opening of business, all personnel are available to handle requests.

You have been asked to perform a preliminary analysis of the technical support environment based on a number of different scenarios. For each scenario, information will be given about the mix of requests and the division of labor among the staff. For each topic, you will given the average number of requests per day for that topic, the average elapsed time before the first of these requests is received, the average time between requests for this topic, and the average time needed to service the request. All times are given in minutes. You will also be given a list of support personnel and, for each one, a list of the topics for which he or she has responsibility. (Since data are based on estimates, factors such as coffee breaks, lunch, computer failures, etc., can be ignored.)

Input

Input consists of a number of scenarios. Each scenario begins with the number of request topics, a positive integer no larger than 20. This is followed by a description of each topic. Each description consists of five integer values: a unique topic identifier, the number of requests for that topic, the elapsed time before the first request for that topic is received, the time needed to service a request, and the time between successive requests. All but the third of these values are positive integers; the elapsed time until the first request could be zero. Following this, the number of personnel is given. This will be a positive integer not to exceed 5. Finally, a description of each person is given in the form of three or more positive integer values: a unique identifying number for the person, the number of topics covered by this person, and a list of the topic identifiers arranged from highest priority to lowest priority for that person. A zero follows the last scenario.

Output

For each scenario, the output consists of the scenario number followed by the statement, *.All requests are serviced within m minutes.,* where m is the number of minutes from the start of the business day until the last request is serviced.

Sample Input

```
3
128 20 0 5 10
134 25 5 6 7
153 30 10 4 5
4
10 2 128 134
11 1 134
12 2 128 153
13 1 153
0
```

Output for the Sample Input

Scenario 1: All requests are serviced within 195 minutes.

Problem 14

Covering Whole Holes

Can you cover a round hole with a square cover? You can, as long as the square cover is big enough. It obviously will not be an exact fit, but it is still possible to cover the hole completely.

The Association of Cover Manufacturers (ACM) is a group of companies that produce covers for all kinds of holes — manholes, holes on streets, wells, ditches, cave entrances, holes in backyards dug by dogs to bury bones, to name only a few. ACM wants a program that determines whether a given cover can be used to completely cover a specified hole. At this time, they are interested only in covers and holes that are rectangular polygons (that is, polygons with interior angles of only 90 or 270 degrees). Moreover, both cover and hole are aligned along the same coordinate axes, and are not supposed to be rotated against each other — just translated relative to each other.

Input

The input consists of several descriptions of covers and holes. The first line of each description contains two integers h and c ($4 \leq h \leq 50$ and $4 \leq c \leq 50$), the number of points of the polygon describing the hole and the cover respectively. Each of the following h lines contains two integers x and y , which are the vertices of the hole's polygon in the order they would be visited in a trip around the polygon. The next c lines give a corresponding description of the cover. Both polygons are rectangular, and the sides of the polygons are aligned with the coordinate axes. The polygons have positive area and do not intersect themselves.

The last description is followed by a line containing two zeros.

Output

For each problem description, print its number in the sequence of descriptions. If the hole can be completely covered by moving the cover (without rotating it), print "Yes" otherwise print "No". Recall that the cover may extend beyond the boundaries of the hole as long as no part of the hole is uncovered. Follow the output format in the example given below.

Sample Input

```
4 4
0 0
0 10
10 10
10 0
0 0
0 20
20 20
20 0
4 6
0 0
0 10
10 10
10 0
0 0
0 10
10 10
10 1
9 1
9 0
0 0
```

Output for the Sample Input

```
Hole 1: Yes
Hole 2: No
```

Problem 15

Code Generation

Your employer needs a backend for a translator for a very SIC machine (Simplified Instructional Computer, apologies to Leland Beck). Input to the translator will be arithmetic expressions in postfix form and the output will be assembly language code.

The target machine has a single register and the following instructions, where the operand is either an identifier or a storage location.

L	load the operand into the register
A	add the operand to the contents of the register
S	subtract the operand from the contents of the register
M	multiply the contents of the register by the operand
D	divide the contents of the register by the operand
ST	store the contents of the register in the operand location

An arithmetic operation replaces the contents of the register with the expression result. Temporary storage locations are allocated by the assembler for an operand of the form "\$n" where n is a single digit.

Input

The input file consists of several legitimate postfix expressions, each on a separate line. Expression operands are single letters and operators are the normal arithmetic operators (+, -, *, /).

Output

Output must be assembly language code that meets the following requirements:

1. One instruction per line with the instruction mnemonic separated from the operand (if any) by one blank.
2. One blank line must separate the assembly code for successive expressions.
3. The original order of the operands must be preserved in the assembly code.
4. Assembly code must be generated for each operator as soon as it is encountered.
5. As few temporaries as possible should be used (given the above restrictions).
6. For each operator in the expression, the minimum number of instructions must be generated (given the above restrictions).

Sample input

AB+CD+EF++GH+++
AB+CD+-

Sample output

L A
A B
ST \$1
L C
A D
ST \$2
L E
A F
A \$2
ST \$2
L G
A H
A \$2
A \$1
L A
A B
ST \$1
L C
A D
N
A \$1

Problem 16

Watch your move!!

We have a square board with 64 places, numbered from 0 to 63, see Figure 1. There are two pieces: A king and a queen. The pair of king's place and queen's place is called the state. A state is legal if the pieces are not at the same place. The king and queen move alternately. The king can move one step in a horizontal or vertical direction, as long as it does not arrive at the place of the queen. The queen can move one or more steps in a horizontal or vertical direction, as long as it does not encounter the king. All these moves are called legal. Please note that the pieces may not move diagonally.

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

Figure 1

For example, suppose we have a state with a king at place 17 and a queen at place 49, as in Figure 2. The legal moves of the king are to places 9, 16, 18, and 25 and the queen can legally move to place 25, 33, 41, 48, 50, 51, 52, 53, 54, 55, or 57. We impose, however, an extra restriction: A piece may not move to a place where the other one can also move to. Legal moves satisfying this restriction are called allowed. In Figure 2, all possible places the king and the queen can move to by an allowed move, are denoted with a circle (o) and a dot (•), respectively. In Figure 3, the king cannot move, it is locked in.

o	K	o					
	•						
	•						
•	Q	•	•	•	•	•	•
	•						

Figure 2.

	Q						
K							

Figure 3.

This problem requires you to write a program that does some checking related to the movement of the queen.

Input

The input for your program resides in a textfile that ends with the standard end-of-file marker. Each line ends with the standard end-of-line marker and contains a sequence of three integers in the range 0..63, separated by one space. The first two integers denote the place of the king and the queen, respectively. Together they form a state. The third integer denotes a new place for the queen. You may think of it as computed by some function `movequeen`. Your program determines whether:

- the given state is legal
- the queen's move is legal
- the queen's move is allowed.

Furthermore, if these requirements are met, your program determines whether the move of the queen results in a state where the king is locked in.

Output

The output is also a text file. For each input line, your program produces one output line with one of the following messages:

- Illegal state
- Illegal move
- Move not allowed
- Continue
- Stop

'Illegal state' indicates that the given state is not legal, i.e. the pieces are at the same place.

'Illegal move' means that the given state is legal, but the queen's move is illegal.

'Move not allowed' applies if both the given state and the queen's move are legal, but the queen's move is not allowed.

Both 'Continue' and 'Stop' mean that the given state is legal and the queen's move is allowed. If the king can do an allowed move in the resulting state, the message is 'Continue', otherwise the king is locked in and you reply 'Stop'.

Sample Input

```
17 17 49
17 49 56
17 49 9
17 49 17
17 49 25
17 49 33
17 49 41
17 49 49
56 48 49
```

Sample Output

```
Illegal state
Illegal move
Illegal move
Illegal move
Move not allowed
Continue
Continue
Illegal move
Stop
```

Problem 17

Morse Mismatches

Samuel F. B. Morse is best known for the coding scheme that carries his name. Morse code is still used in international radio communication. The coding of text using Morse code is straightforward. Each character (case is insignificant) is translated to a predefined sequence of dits and dahs (the elements of Morse code). Dits are represented as periods (".") and dahs are represented as hyphens or minus signs ("-"). Each element is transmitted by sending a signal for some period of time. A dit is rather short, and a dah is, in perfectly formed code, three times as long as a dit. A short silent space appears between elements, with a longer space between characters. A still longer space separates words. This dependence on the spacing and timing of elements means that Morse code operators sometimes do not send perfect code. This results in difficulties for the receiving operator, but frequently the message can be decoded depending on context.

In this problem we consider reception of words in Morse code without spacing between letters. Without the spacing, it is possible for multiple words to be coded the same. For example, if the message "dit dit dit" were received, it could be interpreted as "EEE", "EI", "IE" or "S" based on the coding scheme shown in the sample input. To decide between these multiple interpretations, we assume a particular context by expecting each received word to appear in a dictionary.

For this problem your program will read a table giving the encoding of letters and digits into Morse code, a list of expected words (context), and a sequence of words encoded in Morse code (morse). These morse words may be flawed. For each morse word, your program is to determine the matching word from context, if any. If multiple words from context match morse, or if no word matches perfectly, your program will display the best matching word and a mismatch indicator.

If a single word from context matches morse perfectly, it will be displayed on a single line, by itself. If multiple context words match morse perfectly, then select the matching word with the fewest characters. If this still results in an ambiguous match, any of these matches may be displayed. If multiple context words exist for a given morse, the matching word will be displayed followed by an exclamation point ("!").

We assume only a simple case of errors in transmission in which elements may be either truncated from the end of a morse word or added to the end of a morse word. When no perfect matches for morse are found, display the word from context that matches the longest prefix of morse, or has the fewest extra elements beyond those in morse. If multiple words in context match using these rules, any of these matches may be displayed. Words that do not match perfectly are displayed with a question mark ("?") suffixed.

The input data will only contain cases that fall within the preceding rules.

Input

The Morse code table will appear first and consists of lines each containing an uppercase letter or a digit C, zero or more blanks, and a sequence of no more than six periods and hyphens giving the Morse code for C. Blanks may precede or follow the items on the line. A line containing a single asterisk ("*"), possibly preceded or followed by blanks, terminates the Morse code table. You may assume that there will be Morse code given for every character that appears in the context section.

The context section appears next, with one word per line, possibly preceded and followed by blanks. Each word in context will contain no more than ten characters. No characters other than upper case letters and digits will appear. There will be at most 100 context words. A line containing only a single asterisk ("*"), possibly preceded or followed by blanks, terminates the context section.

The remainder of the input contains morse words separated by blanks or end-of-line characters. A line containing only a single asterisk ("*"), possibly preceded or followed by blanks, terminates the input. No morse word will have more than eighty (80) elements.

Output

For each input morse word, display the appropriate matching word from context followed by an exclamation mark ("!") or question mark ("?") if appropriate. Each word is to appear on a separate line starting in column one.

Sample Input

```
A      .-
B      -...
C      -.-.
D      -..
E      .
F      ...-
G      --.
H      ....
I      ..
J      ,---
K      -.-
L      .-..
M      --
N      -.
O      ---
P      .---.
Q      ---.-
R      .-..
S      ...
T      -
U      ..-
V      ...-
W      ,--
X      -...-
Y      -.---
Z      --..
0      -----
1      ,-----
2      ..---
3      ...--
4      ....-
5      .....
6      -....
7      --...
8      ---..
9      ----.
*
```

```
AN
EARTHQUAKE
EAT
GOD
HATH
IM
READY
TO
WHAT
WROTH
```

```
*
.....
.....
.....
.....
....
-----
*

```

Output for the Sample Input

```
WHAT
HATH
GOD
WROTH?
WHAT
AN
EARTHQUAKE
IM!
READY
TO
IM!
```

Problem 18

Switching Channels

CPN (The Couch Potato Network) owns several cable channels. They would like to arrange the timing of programmes so viewers can switch channels without missing the end of one programme or the beginning of another. To do this they have identified certain times, called "alignment points," where ideally one programme should end and another should begin. Some of these alignment points are more important than others. For example, the time when the nightly news begins is an important alignment point. Since many viewers watch the news, they would be less likely to watch a CPN programme whose ending time causes them to miss the beginning of the news, or which starts before the news finishes. Your task is to write a solution which determines the best order in which programmes can be shown on one channel. A "miss" time is the absolute value of the difference between the time of an alignment point and the nearest time of the beginning or end of a programme. The "total miss time" at a particular level of importance is the sum of all the miss times for alignment points at that level of importance. One programme order is better than another if it has a lower total miss time at some level of importance and the same total miss time at all higher levels of importance (if any).

Input

Your solution must accept multiple input data sets. Each set will begin with an integer, p ($0 < p \leq 8$), specifying the number of programmes to be ordered. When a data set beginning with 0 is encountered, your solution should terminate. Following p on the same line will be p integers specifying the lengths of the programmes in minutes. There is no significance to the order in which these are given.

The next line of input specifies the alignment points. The total number of such points, a ($0 < a \leq 8$), appears first followed by a pairs of integers. The first integer in each pair, i ($1 \leq i \leq 5$), gives the importance of the alignment point. Alignment points marked 1 are most important; those marked 2 are of secondary importance, etc. The second integer in each pair, t , specifies the time when the alignment point occurs. No two alignment points in the same data set will have the same value of t .

Output

Your solution must output three lines for each data set. The first line identifies the data set being processed and should be in the form:

Data set n where n is the number of the data set (1 for the first data set, 2 for the second, etc.). On the following line, your solution should output the lengths of the programmes in the order in which they should be shown to achieve the best synchronization with the alignment points. On the third line, output the total number of minutes by which the alignment points were missed (the sum of all total miss times). There may be more than one best programme order for an input data set. Any one of these best orders is acceptable.

Sample Input

```
4 30 45 45 15
3 1 60 2 90 3 15
6 10 15 13 18 25 33
4 1 30 2 15 2 45 1 60
0
```

Output for the Sample Input

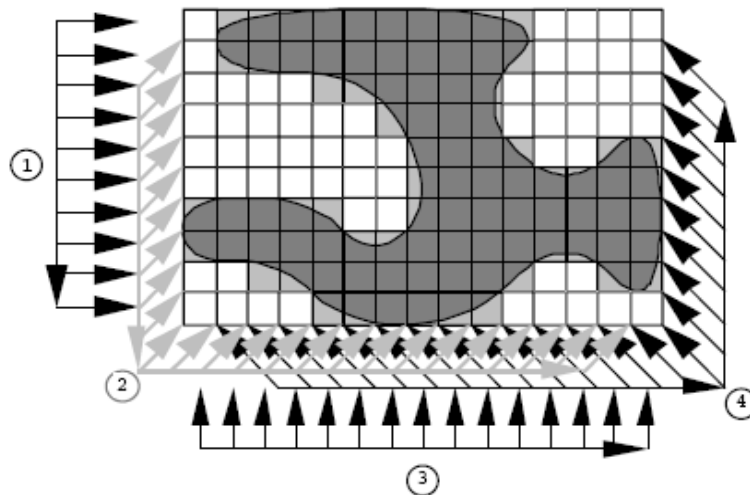
```
Data set 1
Order: 15 45 30 45
Error: 0
Data set 2
Order: 15 13 33 25 18 10
Error: 19
```

Problem 19

Scanner

A body scanner works by scanning a succession of horizontal slices through the body; the slices are imaged one at a time. The image slices can be reassembled to form a three dimensional model of the object. Write a program to construct a two dimensional image slice using data captured during the scan.

The scanner consists of four arrays of sensors arranged around a 10×15 matrix. Array 1 consists of 10 sensors pointing to the right, array 2 has 24 sensors pointing diagonally to the top right, array 3 has 15 sensors pointing to the top and array 4 has 24 sensors pointing to the top left. Each sensor records the thickness of that portion of the object directly in front of that sensor. Readings from the arrays of sensors are recorded in counterclockwise order. Within an array of sensors, data are also recorded counterclockwise. A complete scan consists of 73 readings.



Input

The input file begins with a line with an integer indicating the number of image slices to follow. For each image slice, there are separate lines with 10, 24, 15, and 24 integers representing sensor data from sensor arrays 1 through 4 respectively. The order of the readings is indicated in the diagram. Although it is possible for the result of a scan to be ambiguous, the data supplied to you will have no ambiguous interpretation.

Output

For each slice, your program should print 10 lines of 15 cells. To indicate that the cell represents a part of the object, print a hash character (#) for the cell; to indicate that the cell is not a part of the object, print a period (.). Between successive output image slices, print a blank line.

Sample Input (describing object in diagram above)

```
1
10 10 6 4 6 8 13 15 11 6
0 1 2 2 2 2 4 5 5 6 7 6 5 6 6 5 5 6 6 3 2 2 1 0
2 4 5 5 7 6 7 10 10 10 7 3 3 5 5
0 0 1 3 4 4 4 4 3 4 5 7 8 8 9 9 6 4 4 2 0 0 0 0
```

Output for the Sample Input

```
#####
#####
...#####
...#####
...#####
...#####
#####
#####
#####
#####
#####
#####
```

Problem 20

Crossword Puzzle

Your brilliant but absent-minded uncle believes he has solved a difficult crossword puzzle but has misplaced the solution. He needs your help to reconstruct the solution from a list that contains all the words in the solution, plus one extra word that is not part of the solution. Your program must solve the puzzle and print the extra word.

The crossword puzzle is represented by a grid with ten squares on each side. Figure 1 shows the top left corner of a puzzle. The puzzle has a certain number of “slots” where a word can be placed. Each slot is represented by the row and column number of the square where the slot begins, and the direction in which the slot extends from its initial square (“across” or “down”). The length of each slot is not specified. The puzzle has a list of candidate words, all but one of which is used in solving the puzzle.

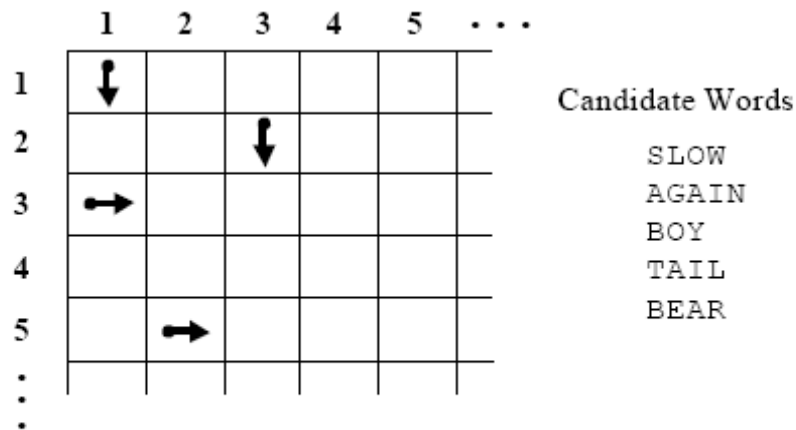


Figure 1: Corner of Example Puzzle

Figure 2 shows a solution to the example puzzle in Figure 1. In a valid solution, each slot is filled with a candidate word. Every maximal horizontal or vertical sequence of two or more letters must be a word in the input. Any candidate word can be used in any slot as long as the word fits in the puzzle and does not conflict with any other word. In the example, all the candidate words are used except the word “BOY”.

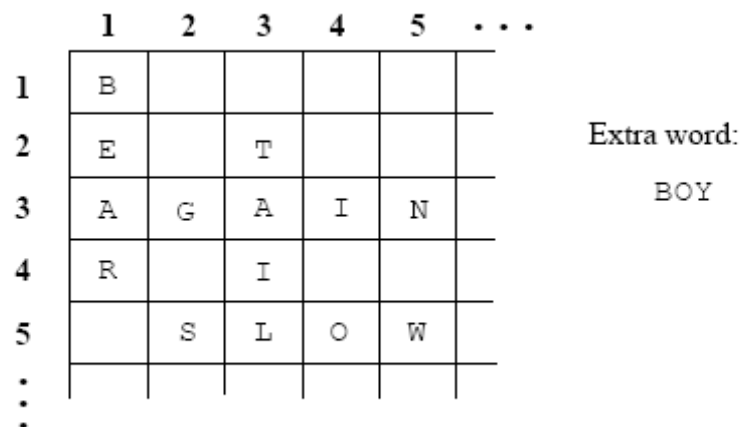


Figure 2: Example Solution

Input

The input data consist of one or more test cases each describing a puzzle trial. The first input line in each test case contains a positive integer N that represents the number of slots in the puzzle. This line is followed by N lines, each containing the row number and column number of a square where a slot begins, followed by the letter 'A' (if the slot is "Across") or 'D' (if the slot is "Down"). The next N + 1 input lines contain candidate words that can be used in the puzzle solution. The final test case is followed by a line containing the number zero.

Output

For each trial, print the trial number followed by the word that is not used in the puzzle solution, using the format in the example output. Observe the following rules:

- 1) Print a blank line after each trial.
- 2) If your uncle has made a mistake and the puzzle has no solution using the given words, print the word "Impossible". For example, if Trial 2 has no solution, you should print "Trial 2: Impossible".
- 3) If the puzzle can be solved in more than one way, print each word that can be omitted from a valid solution. The words can be printed in any order but each word must be printed only once. For example, if Trial 3 has a solution that omits the word DOG and two solutions that omit the word CAT, you should print "Trial 3: DOG CAT" or "Trial 3: CAT DOG".

Sample Input	Output for the Sample Input
4 1 1 D 2 3 D 3 1 A 5 2 A SLOW AGAIN BOY TAIL BEAR 0	Trial 1: BOY

With our best wishes
Dr. Hala Abdel- Galil
Miss. Ensaf Hussein
Miss. Aya Sedky