Problem A: Iterated Difference

Source: `diff.{c,cpp,java}`
Input: `console {stdin,cin,System.in}`
Output: `console {stdout,cout,System.out}`

You are given a list of N non-negative integers a(1), a(2), ... , a(N). You replace the given list by a new list: the k-th entry of the new list is the absolute value of a(k) - a(k+1), wrapping around at the end of the list (the k-th entry of the new list is the absolute value of a(N) - a(1)). How many iterations of this replacement are needed to arrive at a list in which every entry is the same integer?

For example, let N = 4 and start with the list (0 2 5 11). The successive iterations are:

```
2 3 6 11
1 3 5 9
2 2 4 8
0 2 4 6
2 2 2 6
0 0 4 4
0 4 0 4
4 4 4 4
```

Thus, 8 iterations are needed in this example.

Input

The input will contain data for a number of test cases. For each case, there will be two lines of input. The first line will contain the integer N (2 <= N <= 20), the number of entries in the list. The second line will contain the list of integers, separated by one blank space. End of input will be indicated by N = 0.

Output

For each case, there will be one line of output, specifying the case number and the number of iterations, in the format shown in the sample output. If the list does not attain the desired form after 1000 iterations, print 'not attained'.

Sample Input

```
4
0 2 5 11
5
0 2 5 11 3
4
300 8600 9000 4000
16
12 20 3 7 8 10 44 50 12 200 300 7 8 10 44 50
3
1 1 1
4
0 4 0 4
0
```

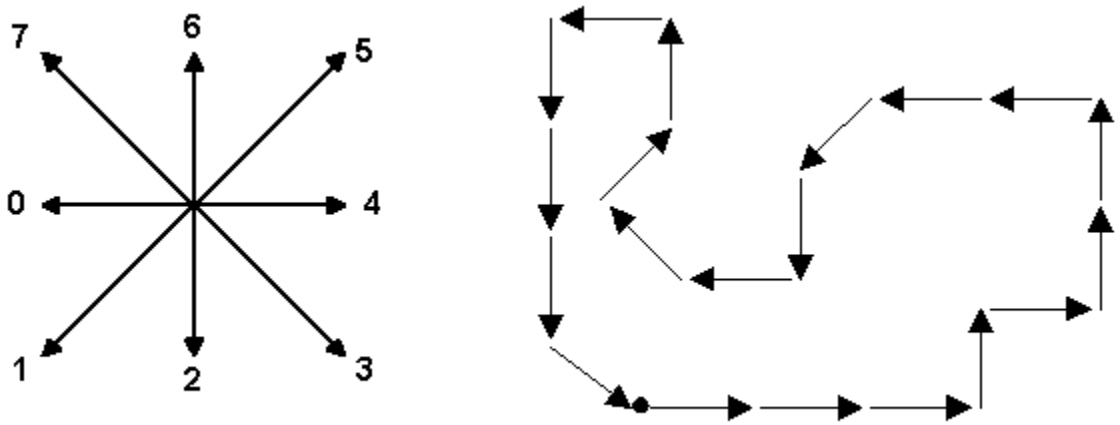
Sample Output

```
Case 1: 8 iterations
Case 2: not attained
Case 3: 3 iterations
Case 4: 50 iterations
Case 5: 0 iterations
Case 6: 1 iterations
```

Problem B: Shape Number

Source: `name.{c,cpp,java}`
Input: `console {stdin,cin,System.in}`
Output: `console {stdout,cout,System.out}`

In computer vision, a chain code is a sequence of numbers representing directions when following the contour of an object. For example, the following figure shows the contour represented by the chain code 22234446466001207560 (starting at the upper-left corner).



Two chain codes may represent the same shape if the shape has been rotated, or if a different starting point is chosen for the contour. To normalize the code for rotation, we can compute the first difference of the chain code instead. The first difference is obtained by counting the number of direction changes in counterclockwise direction between consecutive elements in the chain code (the last element is consecutive with the first one). In the above code, the first difference is

```
00110026202011676122
```

Finally, to normalize for the starting point, we consider all cyclic rotations of the first difference and choose among them the lexicographically smallest such code. The resulting code is called the shape number.

```
00110026202011676122
01100262020116761220
11002620201167612200
...
20011002620201167612
```

In this case, 00110026202011676122 is the shape number of the shape above.

Input

The input consists of a number of cases. The input of each case is given in one line, consisting of a chain code of a shape. The length of the chain code is at most 300,000, and all digits in the code are between 0 and 7 inclusive. The contour may intersect itself and needs not trace back to the starting point.

Output

For each case, print the resulting shape number after the normalizations discussed above are performed.

Sample Input

```
22234446466001207560
12075602223444646600
```

Sample Output

```
00110026202011676122
00110026202011676122
```

Problem C: Stock Prices

Source: `stock.{c,cpp,java}`
Input: `console {stdin,cin,System.in}`
Output: `console {stdout,cout,System.out}`

Buy low, sell high. That is what one should do to make profit in the stock market (we will ignore short selling here). Of course, no one can tell the price of a stock in the future, so it is difficult to know exactly when to buy and sell and how much profit one can make by repeatedly buying and selling a stock.

But if you do have the history of price of a stock for the last n days, it is certainly possible to determine the maximum profit that could have been made. Instead, we are interested in finding the k1 lowest prices and k2 highest prices in the history.

Input

The input consists of a number of cases. The first line of each case starts with positive integers n, k1, and k2 on a line (n <= 1,000,000, k1 + k2 <= n, k1, k2 <= 100). The next line contains integers giving the prices of a stock in the last n days: the i-th integer (1 <= i <= n) gives the stock price on day i. The stock prices are non-negative. The input is terminated by n = k1 = k2 = 0, and that case should not be processed.

Output

For each case, produce three lines of output. The first line contains the case number (starting from 1) on one line. The second line specifies the days on which the k1 lowest stock prices occur. The days are sorted in ascending order. The third line specifies the days on which the k2 highest stock prices occur, and the days sorted in descending order. The entries in each list should be separated by a single space. If there are multiple correct lists for the lowest prices, choose the lexicographically smallest list. If there are multiple correct lists for the highest prices, choose the lexicographically largest list.

Sample Input

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

Sample Output

```
Case 1
1 2 3
10 9
Case 2
8 9 10
2 1
```

Problem D: Contour Tracing

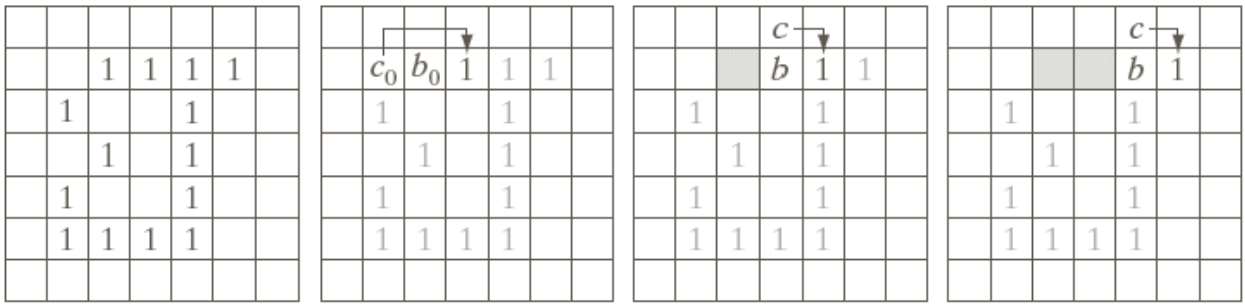
Source: `contour.{c,cpp,java}`
Input: `console {stdin,cin,System.in}`
Output: `console {stdout,cout,System.out}`

In computer vision, objects of interest are often represented as regions of 1's in a binary image (bitmap). An important task in the identification of objects is to trace the contour (also called border and boundary) in an object.

We assume that the bitmap does not contain any 1's on the borders. To trace the contour of a single object, we can use a procedure known as the Moore boundary tracking algorithm as follows:

1. Scan from the top row to the bottom row, from left to right in each row, until an object pixel is found. Call this object pixel b0, and its west background neighbour c0.
2. Examine the 8 neighbours of b0, starting at c0 and proceeding in clockwise direction. Let b1 denote the first neighbour object pixel encountered, and c1 be the background neighbour immediately preceeding b1. Store the locations of b0 and b1, and append b0 and b1 to the contour.
3. Let b = b1 and c = c1
4. Let the 8 neighbours of b, starting at c and proceeding in a clockwise direction, be denoted as n1, n2, ..., n8. Find the first object neighbour nk in this sequence.
5. Let b = nk, c = n(k-1). Append b to the contour.
6. Repeat steps 4 and 5 until b = b0 and the next contour point found is b1. The last two points b0 and b1 are repeated and should not be appended to the contour again.

The first steps of the algorithm is illustrated in the figure below (only the pixels on the boundary are labelled 1 in this bitmap):



In this problem, you will be given a bitmap with at most 200 rows and 200 columns. The bitmap contains a number of objects. You are to determine the length of the contour for each object in the bitmap using the procedure above. In the bitmap, a pixel intensity of 0 represents the background, and a pixel intensity of 1 represents an object pixel. Two pixels with intensity 1 belong to the same object if there is a path from one pixel to another consisting of only 1's, and the path is allowed to follow in any of the 8 compass directions. The borders of the bitmap (first row, last row, first column, last column) contain only background pixels. Any object consisting of fewer than 5 pixels should be ignored as it is most likely noise. None of the objects in the bitmap has holes. Equivalently, there exists a path of background pixels following only the 4 main compass directions (N, S, E, W) for every pair of background pixels in the bitmap.

Input

The input consists of a number of cases. Each case starts with two positive integers on a line, indicating the number of rows (R) and the number of columns (C) in the bitmap. The bitmap is given in the following R lines each containing a string of 0's and 1's of length C. The input is terminated by a case starting with R = C = 0. The last case should not be processed.

Output

For each case, print the case number on its own line. In the next line, print the lengths of the contours of all objects found in the bitmap, sorted in ascending order. Separate the contour lengths by a single space on that line. If the bitmap has no object that have at least 5 pixels, output the line 'no objects found' instead.

Sample Input

```
7 7
0000000
0011110
0111100
0011100
0111100
0111100
0000000
16 7
0000000
0011110
0111100
0011100
0111100
0111100
0000000
0011000
0100110
0000000
0001000
0010100
0010000
0111000
0111000
0000000
4 4
0000
0000
0010
0000
0 0
```

Sample Output

```
Case 1
14
Case 2
8 12 14
Case 3
no objects found
```

Problem E: Pills

Source: `pills.{c,cpp,java}`
Input: `console {stdin,cin,System.in}`
Output: `console {stdout,cout,System.out}`

Aunt Lizzie takes half a pill of a certain medicine every day. She starts with a bottle that contains N pills.

On the first day, she removes a random pill, breaks it in two halves, takes one half and puts the other half back into the bottle.

On subsequent days, she removes a random piece (which can be either a whole pill or half a pill) from the bottle. If it is half a pill, she takes it. If it is a whole pill, she takes one half and puts the other half back into the bottle.

In how many ways can she empty the bottle? We represent the sequence of pills removed from the bottle in the course of 2N days as a string, where the i-th character is W if a whole pill was chosen on the i-th day, and H if a half pill was chosen (0 <= i < 2N). How many different valid strings are there that empty the bottle?

Input

The input will contain data for at most 1000 problem instances. For each problem instance there will be one line of input: a positive integer N <= 30, the number of pills initially in the bottle. End of input will be indicated by 0.

Output

For each problem instance, the output will be a single number, displayed at the beginning of a new line. It will be the number of different ways the bottle can be emptied.

Sample Input

```
6
1
4
2
3
30
0
```

Sample Output

```
132
1
14
2
5
3814986502092304
```

Problem F: Robot Navigation

Source: `robot.{c,cpp,java}`
Input: `console {stdin,cin,System.in}`
Output: `console {stdout,cout,System.out}`

A robot has been sent to explore a remote planet. To specify the path the robot should take, a program is sent each day. The program consists of a sequence of the following commands:

- FORWARD: move forward by one unit.
- TURN LEFT: turn left by 90 degrees. The robot remains at the same location.
- TURN RIGHT: turn right by 90 degrees. The robot remains at the same location.

The robot also has sensor units which allows it to obtain a map of its surrounding area. The map is represented as a grid of M rows and N columns. Each grid point is represented by a coordinate (r,c) where r = 0 is the north edge of the map, r = M-1 is the south edge, c = 0 is the west edge, and c = N-1 is the east edge. Some grid points contain hazards (e.g. craters) and the program must avoid these points or risk losing the robot.

Naturally, if the initial location and direction of the robot and its destination position are known, we wish to send the shortest program (one consisting of the fewest commands) to move the robot to its destination (we do not care which direction it faces at the destination). You are more interested in knowing the number of different shortest programs that can move the robot to its destination, because we may need to send different sequences as interplanetary communication is not necessarily reliable. However, the number of shortest programs can be very large, so you are satisfied to compute the number as a remainder under some modulus, knowing that something you learned in classes called the Chinese remainder theorem can be used to compute the final answer.

Input

The input consists of a number of cases. The first line of each case gives three integers M, N, and the modulus m ($0 < M, N <= 1000$, $0 < m <= 1000000000$). The next M lines contain N characters each and specify the map. A '.' indicates that the robot can move into that grid point, and a '*' indicates a hazard. The final line gives four integers r1, c1, r2, c2 followed by a character d. The coordinates (r1, c1) specify the initial position of the robot, and (r2, c2) specify the destination. The character d is one of 'N', 'S', 'W', 'E' indicating the initial direction of the robot. It is assumed that the initial position and the destination are not hazards. The input is terminated when m = 0.

Output

For each case, print its case number, the modulus, as well as the remainder of the number of different programs when divided by the modulus m. The output of each case should be on a single line, in the format demonstrated below. If there is no program that can move the robot to its destination, output -1 for the number of different programs.

Sample Input

```
3 3 100
***
.*.
***
1 0 1 2 E
4 4 100
****
*.*.
*.*.
*...
1 1 1 3 N
4 8 100
********
...**...
*......*
********
1 0 1 7 E
0 0 0
```

Sample Output

```
Case 1: 100 -1
Case 2: 100 2
Case 3: 100 4
```

Problem G: User Names

Source: `username.{c,cpp,java}`
Input: `console {stdin,cin,System.in}`
Output: `console {stdout,cout,System.out}`

A university's computer system assigns user names according to the following set of rules:

1. The maximum length of a username is MAXLEN characters. (The value of MAXLEN will be specified in the input for each problem instance.)
2. The first character of the user name is the first letter of the person's first name, converted to lower case. Ignore apostrophes and hyphens here and in Step 3.
3. Append as many letters of the person's last name as possible (converted to lower case, if necessary), without exceeding a total of MAXLEN characters. Starting with the first letter of the last name, append these letters in the order in whch they appear in the last name.
4. If a user name assigned on basis of Rules 1 - 3 already exists in the database, break the tie as follows: append serial numbers 1 - 9, in that order, to the username from step 3, if that can be done without exceeding the limit of MAXLEN characters in the username. Otherwise, drop the last letter before appending the serial number.
5. If a user name assigned on basis of Rules 1 - 4 already exists in the database, break the tie as follows: append serial numbers 10 - 99, in that order, to the username from step 3, if that can be done without exceeding the limit of MAXLEN characters in the username. Otherwise, drop the last letter or the last two letters (whichever is necessary) before appending the serial number.
6. It is assumed that the above rules will avoid ties.

Input

The input will contain data for a number of test cases. The first line of each test case will contain two positive integers: the number of names and the value of MAXLEN (5 <= MAXLEN <= 80). This will be followed by the list of names. Each name will consist of at most 80 characters and will begin with the first name, followed by middle names, if any, and will conclude with the last name. A single blank space will separate first, middle, and last names. Any name can contain upper and lower case letters, hyphens, and apostrophes. A last name will contain at least two letters, other names will contain at least one letter (they could be just initials). There will be no more than 200 names in each case. The last test case will be followed by a line containing two zeros for the number of names and MAXLEN.

Output

For each case, the output will begin with a line containing the case number. This will be followed by the list of user names, one per line, in the same order as the corresponding names in the input.

Sample Input

```
2 6
Jenny Ax
Christos H Papadimitriou
11 8
Jean-Marie d'Arboux
Jean-Marie A d'Arboux
Jean-Marie B d'Arboux
Jean-Marie C d'Arboux
Jean-Marie D d'Arboux
Jean-Marie D d'Arboux
Jean-Marie F d'Arboux
Jean-Marie G d'Arboux
Jean-Marie H d'Arboux
Jean-Marie I d'Arboux
Jean-Marie J d'Arboux
11 9
Jean-Marie d'Arboux
Jean-Marie A d'Arboux
Jean-Marie B d'Arboux
Jean-Marie C d'Arboux
Jean-Marie D d'Arboux
Jean-Marie D d'Arboux
Jean-Marie F d'Arboux
Jean-Marie G d'Arboux
Jean-Marie H d'Arboux
Jean-Marie I d'Arboux
Jean-Marie J d'Arboux
0 0
```

Sample Output

```
Case 1
jax
cpapad
Case 2
jdarboux
jdarbou1
jdarbou2
jdarbou3
jdarbou4
jdarbou5
jdarbou6
jdarbou7
jdarbou8
jdarbou9
jdarbo10
Case 3
jdarboux
jdarboux1
jdarboux2
jdarboux3
jdarboux4
jdarboux5
jdarboux6
jdarboux7
jdarboux8
jdarboux9
jdarbou10
```

Problem H: Citizenship Application

Source: `citi.{c,cpp,java}`
Input: `console {stdin,cin,System.in}`
Output: `console {stdout,cout,System.out}`

In Canada, once you have landed as a permanent resident, you can apply for citizenship if you have lived in Canada for at least 1095 days (approximately 3 years) in the 1460 days (approximately 4 years) immediately prior to the application for citizenship (i.e. the date of application is not counted). Any time travelling outside Canada is not counted as a day living in Canada. Furthermore, if you are already residing in Canada (e.g. to study) before landing as a permanent resident during the 1460-day period, that time in Canada before the landing date (up to 730 days) can be counted as half rounded down (e.g. if the waiting time was 101 days, that can be counted as 50 days). Thus, the starting date of residence is counted as a half day if it occurs before the landing date (assuming that it is not more than 730 days before), or a full day if it coincides with the landing date.

In this problem, you will be determining the first day on which an application for citizenship can be made.

Input

The input consists of a number of cases. The first line of each case gives the starting date of residence in the country, and the second line gives the landing date as a permanent resident. The third line gives an integer N (0 <= N <= 100) indicating the number of travels outside of Canada. Each of the next N lines contains two dates separated by a space, indicating the start and end date (inclusive) of travel outside of Canada. That is, you are considered to be outside of Canada from the start date through the end date.

You may assume that the starting date of residence is no later than the landing date. You may also assume that the start date of each travel is no later than the end date, and no travel outside of Canada will be longer than 200 days. Of course, you can assume that your trips do not include the starting date of residence and the landing date, and no two trips overlap. No trips take place before the starting date of residence. All dates are given in the form Month/Day/Year and are valid dates, and no dates in the input will be before January 1, 1980 or after December 31, 2020.

Output

For each case, print on one line the date of the first day (in the same format as the input) on which an application for citizenship can be made.

Sample Input

```
1/1/2011
9/1/2011
0
1/1/2011
9/1/2011
1
10/1/2011 10/10/2011
1/1/2011
9/1/2011
2
2/1/2011 3/28/2011
10/1/2011 10/10/2011
1/1/2009
1/1/2011
0
12/31/2020
12/31/2020
0
```

Sample Output

```
5/2/2014
5/12/2014
6/9/2014
12/31/2012
12/31/2023
```

Problem I: Wealthy Family

Source: `family.{c,cpp,java}`
Input: `console {stdin,cin,System.in}`
Output: `console {stdout,cout,System.out}`

While studying the history of royal families, you want to know how wealthy each family is. While you have various 'net worth' figures for each individual throughout history, this is complicated by double counting caused by inheritance. One way to estimate the wealth of a family is to sum up the net worth of a set of k people such that no one in the set is an ancestor of another in the set. The wealth of the family is the maximum sum achievable over all such sets of k people. Since historical records contain only the net worth of male family members, the family tree is a simple tree in which every male has exactly one father and a non-negative number of sons. You may assume that there is one person who is an ancestor of all other family members.

Input

The input consists of a number of cases. Each case starts with a line containing two integers separated by a space: N (1 <= N <= 150,000), the number of people in the family, and k (1 <= k <= 300), the size of the set. The next N lines contain two non-negative integers separated by a space: the parent number and the net worth of person i (1 <= i <= N). Each person is identified by a number between 1 and N, inclusive. There is exactly one person who has no parent in the historical records, and this will be indicated with a parent number of 0. The net worths are given in millions and each family member has a net worth of at least 1 million and at most 1 billion.

Output

For each case, print the maximum sum (in millions) achievable over all sets of k people satisfying the constraints given above. If it is impossible to choose a set of k people without violating the constraints, print 'impossible' instead.

Sample Input

```
5 3
0 10
1 15
1 25
1 35
4 45
3 3
0 10
1 10
2 10
```

Sample Output

```
85
impossible
```

Problem J: Supply Mission

Source: `supply.{c,cpp,java}`
Input: `console {stdin,cin,System.in}`
Output: `console {stdout,cout,System.out}`

You are to fly a helicopter to bring supplies to a number of submarines that are currently moving across the ocean. You will be given the coordinates of the helicopter base and each submarine. Each submarine is travelling constantly at a heading and speed specified by the velocity vector (vx, vy), meaning that after 1 hour it would have travelled vx km in the x direction and vy km in the y direction (vx and vy may be negative). The length of the vector is the speed. The helicopter is capable of travelling at a constant speed in any direction (assume that acceleration and deceleration are instantaneous). The helicopter must land on each submarine at least once, and it takes 1 hour at each stop to unload the supplies and refuel. Each submarine rises to the surface at the appropriate landing time, and submerges once the helicopter leaves. You may assume that the velocity of each submarine is not affected by any change of its travelling depth. The helicopter can carry enough supplies for all submarines without having to return to the base. All coordinates have km as units, and all velocities and speeds have km/h as units.

Find the shortest time for the helicopter to bring supplies to each submarine from the base and return to its base.

Input

The input consists of a number of cases. Each case starts with a line containing the integer N ($1 \le N \le 8$) specifying the number of submarines. The next N lines contain 4 integers separated by a space: the initial (x,y) coordinates of the i-th submarine and its velocity vector. The last line of each case contains 3 integers specifying the (x,y) coordinates of the helicopter base and the speed of the helicopter. The end of input is indicated by a case that starts with $N = 0$, and this last case should not be processed. All input integers have absolute value at most 1000. You may assume that the helicopter travels at a greater speed than every submarine. Note that the paths of the submarines may cross each other or even the helicopter base, but since they can adjust their depth there will be no collisions.

Output

For each case, print its case number, a colon, followed by the minimum amount of time needed to complete the mission in the format:

```
Case a: b hour(s) c minute(s) d second(s)
```

where a, b, c, d are appropriate non-negative integers, and c and d are at most 59. The time should be rounded up to the next second.

Sample Input

```
5
1 0 0 0
2 0 0 0
3 0 0 0
4 0 0 0
5 0 0 0
0 0 1
3
1 2 3 4
2 2 40 23
7 8 22 10
0 0 50
0
```

Sample Output

```
Case 1: 15 hour(s) 0 minute(s) 0 second(s)
Case 2: 5 hour(s) 59 minute(s) 50 second(s)
```