Problem A: Two-Way Carry Propagation

In 1897, the mathematician L. Aguile invented a special operation (#) over the binary representation of integer numbers. In its simple form, $A\#B$ is computed according to the following steps, where A is a non-negative integer number and B is of the form 2^k for some integer k ($0 \le k \le 7$):

1. Consider the 8-bit binary representation of the numbers A and B, and name them A' and B' respectively.
2. Compute $C = A' + B'$ in base 10 (i.e. 1+1=2). Assume $\overline{c_7c_6c_5c_4c_3c_2c_1c_0}$ be the sequence of digits in C.
3. Since the addition is done in base 10, there may be some digit $c_i = 2$. For such a digit, change c_i to 0, and add 1 to c_{i-1} and c_{i+1}. In case $i = 0$, only add 1 to c_{i+1}. You may assume that the input numbers are small enough that this case never happens for $i = 7$.
4. The step 3 is repeated until there is no digit 2 in C, which is finally considered as the binary representation of $A\#B$.

For example, if A = 23 (binary 00010111), and B = 2 (binary 00000010), then the following sequence of numbers defines the value of C in successive stages of the above algorithm: 00010121 → 00010202 → 00010210 → 00011020 → 00011101 which is the number 29.

The problem is to input A and B, and output $A\#B$. All numbers are in expressed in base 10 and you must take care of the conversions to binary and vice versa.

Input (Standard Input)

The input consists of several test cases. Each test case comes on a separate line containing two integer numbers A and B, separated by blanks, where A is between 0 and 255, and $B = 2^k$ for some $0 \le k \le 7$. You may assume that at every step during the computation of $A\#B$ as defined above, C fits in an 8-bit number. You may assume that there are no blank characters at the beginning or at the end of the lines. The input terminates with a single line containing two zero which should not be processed.

Output (Standard Output)

For each test case, output a single line containing the number $A\#B$ in decimal.

Sample Input and Output

Standard Input	Standard Output
23 2 7 1 64 16 0 0	29 11 80

Problem B: Calculating Flight Duration

Naaah! Yet another time you receive a flight ticket and have to calculate how long your flight takes using confusing format that the airlines use to describe the departure/arrival times. But this time, you decide to solve this problem once and for all by writing a program to calculate the flight duration! For simplicity, in the first version of the problem you only handle non-stop flights.

Input (Standard Input)

The input contains several test cases. Each test case describes a non-stop flight on a single line of the input. Each flight description begins with flight number (a string of at least 2 at most 7 alphanumeric characters) followed by the departing city name, time zone, and the departure time and also the arriving city name, time zone and the arrival time. City names are strings of at least 1 at most 9 alphanumeric characters.

The time zone is written as a number inside a pair of parentheses with the format shown in the sample input, like `(+3.5)`. Note that the plus/minus sign is always present. This number specifies the offset from the UTC standard time, expressed in hours (so a time in the time zone +3.5 is three hours and half more than the time in UTC). There is no space character between the city names and the time zones.

The flight times are described in the format shown in the sample input, e.g., `4:10a`. Note that the time is presented in 12-hour format, the minutes is a two-digit number. A character 'a' or 'p' immediately follows the time indicating AM and PM times respectively. In case a flight arrives the next day, a '+1' is appended to the arrival time. Assume that no flight takes more than 24 hours (hence we do not have a +2 symbol in arrival times), and there is no test case in which the flight arrives the day before (hence we do not have a -1 symbol in arrival times).

The end of input is specified by a single line containing a single # character which should not be processed.

Output (Standard Output)

For each test case, output a single line containing the flight number, one space character, and the duration of the flight. The duration must be of form <hours>:<minutes>, where <minutes> is formatted as a two-digit number.

Sample Input and Output

Standard Input	Standard Output
EK976 Tehran(+3.5) 4:10a Dubai(+4) 6:45a EK432 Dubai(+4) 8:40a Singapore(+8) 7:40p JL710 Singapore(+8) 11:20p Tokyo(+9) 7:00a+1 EK0215 Dubai(+4) 8:20a LA(-8) 12:50p #	EK976 2:05 EK432 7:00 JL710 6:40 EK0215 16:30

Problem C: Pausotopic messages

Due to the increased availability of mobile phones and the decreased cost of short message services (SMS), text messaging is getting more and more popular. To do text messaging, one of course needs a text entry system in his mobile phone. One well-known text entry system is multi-tap in which letters are printed on 8 keys in either 3-letter or 4-letter sequences as depicted in the figure. The system is used by repeatedly pressing the same key to cycle through the letters for that key. For example, pressing the key "4" twice would indicate the letter "H". Pausing more than a threshold time automatically chooses the current letter in the cycle, as pressing a different key. If someone is not careful about the threshold time, he may enter a wrong text due to long or short unwanted pauses in pressing keys. For example, if he wants to type "C", the key "2" must be pressed 3 times but long unwanted pauses while pressing key "2" may produce "AAA", "AB" or "BA". As another example, if he wants to type "QP", a short pause after the second pressing of key "7" may produce "R".

Two messages are called pausotopic if and only if making pauses shorter or longer while typing one of the two given messages produces the other message. Your task is to determine whether the two given messages are pausotopic. You may assume each letter in both messages is typed with the minimum number of pressing, i.e., a letter is not typed with cycling letters on a key several times For instance, although "U" can be typed by pressing "8" five times, this assumption says "U" is typed with pressing "8" twice.

Input (Standard Input)

The input consists of several test cases. The first line of the input consists of a single integer n which is the number of test cases. Each test case consists of two words given in two consecutive lines. Each word is a string of capital letters (A-Z) with size at least 1 and at most 150.

Output (Standard Output)

For each test case, in a single line output "YES" if the given two words are pausotopic; otherwise, output "NO".

Sample Input and Output

Standard Input	Standard Output
3 QQ PS AAAAA CC ABC CBA	NO NO YES

Problem D: AutoComplete Tool

As ACM programming contest in Tehran site is getting closer and closer, MIT team is training more and more to finally get qualified for the final contest. All three members of MIT team are now expert in designing and implementing algorithms even complicated ones but they suffer from slow typing. Although they have been typing for many years, their typing speed is the same as that of a beginner programmer. This problem, together with a limited time in an ACM programming contest, has been resulted in their obtaining a world record of getting no balloons while attending the most ACM contests.

To reach the dream of attending the final contest, they have decided to develop a software, named AutoComplete Tool (ACT for short), to expedite their coding. ACT aims to reduce the number of key-pressings needed to type a code (or a text) by predicting words. For a text to be typed, assume that ACT knows all words in the text. ACT constructs a mapping table which maps a string s to a word (denoted by $w(s)$) of the text where s is a non-empty prefix of $w(s)$. ACT uses the mapping table to suggest $w(s)$ to a user when s is being typed by the user. ACT suggestions are consistent, i.e., if the suggested word of s is $w(s)$ and ss' (the concatenation of two strings s and s') is a prefix of $w(s)$, then $w(ss') = w(s)$. When a user starts typing the text, ACT helps the user as follows. Upon typing a letter, ACT suggests $w(s)$ where s is the string of letters being typed so far. Pressing "enter" key ("↵") results in displaying the suggested word without typing any more letters. In the case of pressing "space" key ("␣"), the word being typed so far is displayed. In both cases, a space is displayed after the displayed word. For instance, assume the text is "hello heat heats ". In the mapping table, suppose ACT maps "h" and "hel" to "heats" and "hello" respectively. Pressing the keys "h", "e", "l", "↵", "h", "e", "a", "t", "␣", "h", "↵" in the given order displays the text. Note that the suggested word of "h", "he", "hea", "heat" and "heats" is "heats". Assuming ACT constructs the mapping table in such a way that the number of key-pressings is minimum, your task is to compute the minimum number of key-pressings using ACT for typing a text.

Input (Standard Input)

The input consists of several test cases. Each test case begins with an integer n $(0 < n < 10000)$ denoting the number of words in the text. Each of the next n lines contains a word. Words are strings of lowercase letters (a..z) of size at least 1 and at most 15. The input terminates with a line containing 0.

Output (Standard Output)

For each test case, output a single line containing the minimum number of key-pressings needed to type all words of the text when an ACT is available. Note in the text, words are separated by a space and there is a space after the last word.

Sample Input and Output

Standard Input	Standard Output
1 hello 3 heat hello heats 2 da dc 2 abc abc 0	2 11 5 4

Problem E: Training Private Ryan

Private Ryan is taking the training of his military service. As part of the training, he participates in a session where he must shoot a number of targets quickly using an M-4 rifle. Each target appears at a specific moment of time and will disappear at some moment in future. Furthermore, each target is assigned a specific score, and Ryan can shoot a target more than once. There is a reloading time L for the rifle, so there should be at least L time steps between any consecutive shots. When he decides to shoot, he can choose one of the targets being present at the moment. What is the biggest score Ryan can achieve, assuming that he can use at most n bullets?

Input (Standard Input)

There are multiple test cases in the input. Each test case starts with 3 numbers n (the number of bullets), L (the rifle reloading time), and k (the number of targets) in a line. The following k lines contain description of the targets. Each target is described by three numbers; appearing time (ta), disappearing time (td) and its score (s). Ryan can shoot the target in the closed interval $[ta, td]$ with $ta < td$. Note that all of the input numbers are non-negative integers, where $n \leq 100$ and $L \leq 10^9$, and for each target, $0 \leq ta < td \leq 10^9$. The scores are not greater than 1,000,000 and $k <= 1000$. The last line of the input contains three zeros which should not be processed.

Output (Standard Output)

For each test case, write a single line in the output containing the maximum possible score that Ryan can achieve.

Sample Input and Output

Standard Input	Standard Output
4 2 3 2 7 8 1 3 6 0 12 3 2 10 3 1 2 3 2 10 5 10 11 3 0 0 0	30 6

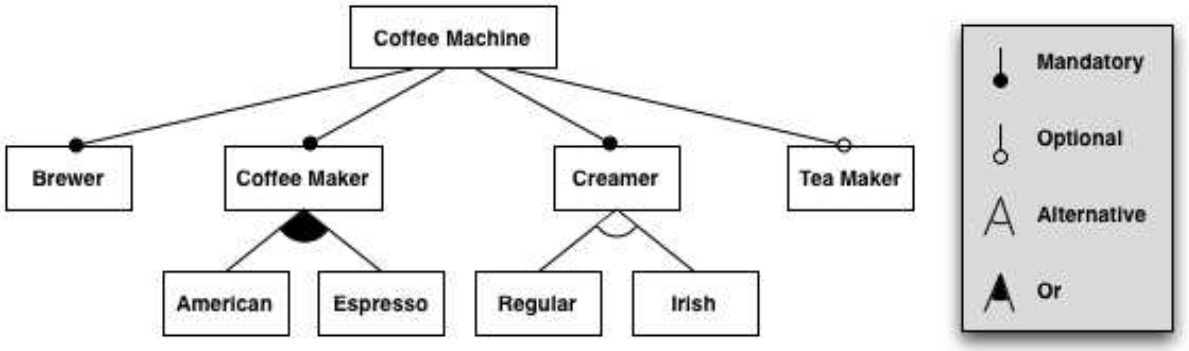
Problem F: Advanced Coffee Makers

Advanced Coffee Makers (ACM) Company holds a large portion of the coffee vending machine market. Facing with numerous variations of coffee machines that the market demands for, ACM tries to develop a product line, so that each time a customer requests for a specific configuration of coffee machines, it can easily produce and deliver the product in a shorter time and with lower cost. This way, if a customer asks for example, for a coffee machine that serves both coffee and tea, but does not offer options for adding cream to coffee, ACM can produce the corresponding machine including coffee and tea features, but not having creamer features.

To this end, ACM has prepared a feature model of the coffee machine products. The feature model contains a hierarchy of the features appearing in a coffee machine, organized as a rooted tree. The root always corresponds to the whole coffee machine product, while other nodes represent the features of a coffee machine. Relationships between a parent feature and its child features (or sub-features) are categorized as:

- Mandatory – child feature is required.
- Optional – child feature is optional.
- Or – at least one of the sub-features must be selected.
- Alternative (xor) – exactly one of the sub-features must be selected.

For example, the following figure represents a sample feature model for coffee machines:



In the above figure, every coffee machine must have Brewer, Coffee Maker, and Creamer component. However, it may or may not have a Tea Maker component. It can provide American or Espresso types of coffee, or both. But, the Creamer can be either Regular or Irish.

Having a feature model, a customer may ask for a specific configuration, i.e., a set of features. For example, {Coffee Machine, Brewer, Coffee Maker, American, Creamer, Regular} form a valid configuration. Note that mandatory features must be present in a valid configuration if their parents are present. In contrast, the configuration {Coffee Machine, Brewer, Coffee Maker, Creamer, Regular, Tea Maker} is not valid, since no sub-feature of Coffee Maker is selected. Note that the whole Coffee Machine (the root) must be included in all valid configurations.

A valid configuration must include the root feature. A mandatory feature must be included in the configuration if its parent is included. From a set of features having ‘or’ (resp. ‘xor’) relationship with an included parent feature, at least (resp. exactly) one must be present. Also, if a child is included in a valid configuration, its parent must also be included. You may assume that each feature appears only once in the feature model tree.

Your program must input a feature model, together with a set of configurations, and for each configuration determines whether the configuration is valid.

Input (Standard Input)

The input consists of a number of test cases. Each test case has two parts. The first part represents the feature model and the second part lists the configurations to be validated. The feature model is described as a set of lines of the following forms, each describing a (non-leaf) feature F that is composed from a set of $n > 1$ subfeatures:

- $F = F_1 + F_2 + \cdots + F_n$, where F_i is either a feature name, or a question mark followed by a feature name.
- $F = F_1 \mid F_2 \mid \cdots \mid F_n$, where F_i is a feature name
- $F = F_1 \text{ ^ } F_2 \text{ ^} \cdots \text{ ^ } F_n$, where F_i is a feature name

Each line defines the sub-features of a feature F. In the first case, optional features are preceded by a question mark (see the sample input). It is assumed that the first line defines the whole coffee machine (that must be present in all valid configurations). The feature names are sequences of upper-case and lower-case letters (with no blanks in between). However, there may be arbitrary number of blank characters in the beginning or at the end of the lines, or around the symbols =, +, ?(Optional), |(Or), and ^(Alternative). You may safely assume that each feature is defined once, and the feature model forms a tree. As said before, each feature appears only once in the feature tree (e.g. we cannot have A=B+C and B=C+D). Also assume that the total number of features is at most 1000 (and of course at least one).

After the last line of a feature model description there is a line containing a single # character, after which come the lines describing the configurations that you must validate. Each configuration comes in a separate line of the form $\{F_1, F_2, \ldots, F_n\}$, where F_i is a feature name and $n > 0$. There may be blank characters anywhere in the line except in the feature names. You may assume that each F_i has appeared in the feature model description and there are no duplicate feature names in a configuration. The last line of the test case is a single line containing ##. After the last test case, there is a line containing ### which should not be processed.

Output (Standard Output)

For each configuration in each test case, print one line in the output containing either Valid or Invalid words, indicating whether the corresponding configuration is valid. Print a line containing +++ (three plus sign) after each test case.

Sample Input and Output

Standard Input	Standard Output
CoffeeMachine = Brewer + CoffeeMaker + Creamer + ?TeaMaker CoffeeMaker = American | Espresso Creamer = Regular ^ Irish # {CoffeeMachine, Brewer, CoffeeMaker, American, Creamer, Regular} {CoffeeMachine, Brewer, CoffeeMaker, Creamer, Regular, TeaMaker} ## A=?B+?C B=D|E # {A,B,C,D,E} {A,D} ## ###	Valid Invalid +++ Valid Invalid +++

Problem G: The Security Team

An important welcome ceremony of some political officials is going to be held and as the manager of the security team, you are going to put your n agents along the red carpet. The agents are numbered 1 through n and equipped with wireless monaural headsets. The authorizations of the headsets are configured such that agent $\#i$ can only talk to agents $\#i-1$ and $\#i+1$ (if they exist of course). In addition, the headset of agent $\#i$ has a wireless range r_i, and agents $\#i$ and $\#j$ can talk only if their distance does not exceed the minimum of r_i and r_j. Your number-one rule of security is that any two agents should always be able to communicate directly or through some intermediate agents.

The red carpet is laid on a straight line, and the longer the carpet is, the more magnificent the ceremony looks like. But, the length of the carpet is limited since there must always be an agent on each head of the carpet and the agents must be connected as your number-one rule of security states.

Everything was running well till your boss arrived at the rehearsal of the ceremony one day before the main event. Having a look on your agents, the boss said:

> "The current placement of the agents along the red carpet does not seem so graceful. Arrange them along the carpet in ascending order of their heights. And by the way, the distance of no two agents should be less than one meter."

Confused from the new orders of the boss, you figure out that there is no time to reconfigure the authorization of the agents' headsets, and the only way to apply the new orders while keeping your own connectivity rule, is to shorten the length of the red carpet. Given the wireless range of the headsets and the height ordering of the agents, your job is to find the maximum possible length of the red carpet that could be used in the ceremony.

Input (Standard Input)

There are multiple test cases in the input. Each test case starts with a line containing the integer n ($1 \le n \le 10^4$), followed by a line having n space-separated integers which lists the agents' numbers in ascending order of their heights (e.g. 3 1 2 indicates agent #3 is the shortest and agent #2 is the tallest). It is guaranteed that all numbers 1 through n appear exactly once in this line. The test case ends with a line containing n space-separated integers $r_1\ r_2\ \ldots\ r_n$ where r_i ($1 \le r_i \le 10^5$) is the wireless range of the headset of agent $\#i$ given in meters. The input ends with a line containing a single zero which should not be processed as a test case.

Output (Standard Output)

For each test case, you should print the maximum possible length of the red carpet along which you can put your agents (with an agent on each end) and keep all the agents connected while obeying the rules of the boss. If it is not possible to satisfy all the requirements together, write "`Impossible`" instead (without quotation marks).

Sample Input and Output

Standard Input	Standard Output
5 3 4 5 1 2 6 6 6 6 6 3 1 3 2 1 2 1 0	6 Impossible

Problem H: Mathematics Test

A mathematics teacher wants to test her students one by one, on basic math. The teacher uses N balls in the test, on each a single integer is written. In order to test one student (say the i^{th} student), the teacher first divides all the N balls into K_i non-empty groups, randomly. Then, she asks the student to tell the number of balls and the maximum number written on the balls, for each group. It is supposed to be K_i pairs of integers, considered as the student's answer.

As an observer from the bureau of education, you are present in the classroom! Initially all you know is N (the number of balls used in the test) and M (the number of students who are taking the test). Later, during the test for student i ($1 \le i \le M$) you will hear an integer K_i given by the teacher, along with K_i pairs of integers as the student's answer. These pair of integers are in the format of $\langle a_j, b_j \rangle$ each, where a_j and b_j are the number of balls and the maximum number for the j^{th} group respectively ($1 \le j \le K_i$).

Since you are in love with logical problems, you are curious to figure out whether it is possible that all of these answer be correct using the same set of balls or not!

Input (Standard Input)

You are visiting several classes (which means "there are multiple test-cases" in ACM-ICPC lingo!) For each class, you are given two integers N ($1 \le N \le 1000$) and M ($1 \le M \le 1000$) in the first line. In the next M lines, the description of each test comes. Each line starts by K_i, the number of groups, followed by $2 \times K_i$ space-separated numbers $a_1\ b_1\ a_2\ b_2\ \ldots\ a_{K_i}\ b_{K_i}$ where a_j is the number of balls in the j^{th} group and b_j is the maximum number in that group. You may assume all these numbers are positive integers less than or equal to 1,000,000,000.

The last class is empty ($N = 0$ and $K = 0$) and it should not be processed.

Output (Standard Output)

For each class, the output should be "Impossible" if you have enough evidences (and you are 100% sure) that the students' answers cannot be all correct at the same time and at least one of them must be wrong. Otherwise, it must be "Possible", where there is a chance the class did well altogether! Write the output without quotation marks.

Sample Input and Output

Standard Input	Standard Output
3 2 2 1 10 2 4 2 1 6 2 10 10 2 2 5 5 5 10 3 3 3 3 6 4 10 0 0	Impossible Possible

Problem I: GPS Reverse Positioning

It's a misty day in Manhattan and you have lost your way. Driving your car, you are trying to figure out what your current location is. The foggy weather does not allow you to see any sign or information outside the car, and like other nerds, you hate asking for directions. But fortunately enough, you already have two other sources of information: a map of Manhattan, and a GPS/navigational module installed on your car. The bad news is that the module is malfunctioning due to high humidity and it does not accept any user inputs. Therefore, you can only use the information available on its current screen, which is a list of locations you had already entered on the system (called points of interest) each with the shortest distance (through the roads) from your current position to that location.

Given the map data and the information seen on the screen, your job is to find all locations on the map which are possibly your current position.

The map is modeled with N map nodes, and R two-way roads. Each node is a point on the map specified by its coordinates. Each road is a line segment on the map specified by the two nodes it connects, which are called its heads. As you might already know, all roads in Manhattan are either vertical or horizontal. A node might be the head of one or more roads. A car can go from a road r to a node n and vice versa (i.e. from n to r), if and only if the node n is a head of the road r. By definition, it is not possible to drive a car from a road directly to another road without using any intermediate nodes, even if the roads have a nonempty intersection, as the city has many non-planar structures. Similarly, you cannot drive from one node directly to another node without using any intermediate roads even if they have the same coordinate on the map. Though, it is guaranteed that you can drive from any node of the map to any other node using some intermediate roads and nodes.

The points of interest seen on the screen are also given as some nodes of the map. Though, your current position is not necessarily on a node of the map; in this case, you are on a road. The shortest-path distances given on the screen are calculated according to the rules specified above.

Input (Standard Input)

The input contains several test cases. Each test case starts with a line containing three space-separated integers N ($1 \le N \le 10000$), the number of nodes, R ($1 \le R \le 10000$), the number of roads, and M ($1 \le M \le 100$), the number of the points of interest. The nodes are numbered $1, 2, \dots, N$. The next N lines specify the coordinates of the nodes. The i^{th} line of this part contains two space-separated integers x_i and y_i ($-10^5 \le x_i, y_i \le 10^5$), the coordinates of node i. Then, R lines follow, each with two space-separated integers u_i and v_i, indicating that the two nodes u_i and v_i are connected by a road. It is guaranteed that the road is either vertical ($x_{u_i} = x_{v_i}$) or horizontal ($y_{u_i} = y_{v_i}$). The test case ends with M lines specifying the information available on the GPS system screen. Each of these lines has two space-separated integers w_i and d_i ($0 \le d_i \le 10^9$), where w_i is the number of a node as a point of interest, and d_i is the distance from your current position to node w_i.

The input ends with a line of the form "`0 0 0`" (omit the quotes) which should not be processed as a test case.

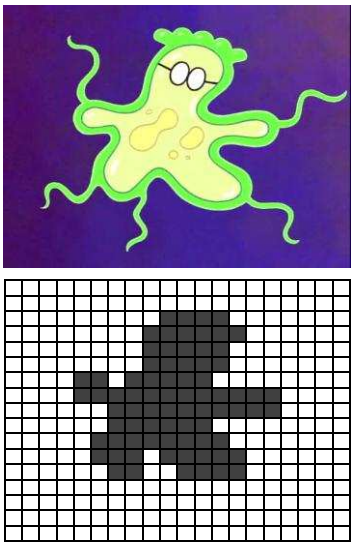
Output (Standard Output)

For each test case, you should first write a line containing a single integer k, the number of distinct points on the map that might be your current position. Then, on each of the next k lines, you should write two space-separated integers X_i and Y_i as the coordinates of one of the possible points. The coordinates must be in ascending order based on their X_i's, and then by their Y_i's (when their X_i's are equal).

Sample Input and Output

Standard Input	Standard Output
7 6 2 0 1 1 0 1 1 1 3 3 1 3 3 4 0 1 3 3 5 5 6 4 3 3 2 2 7 7 5 5 3 3 2 1 0 0 2 0 3 0 1 2 1 3 1 1 3 2 1 0 0 0 2 0 0 1 2 2 3 1 3 3 2 2 0 0 0 2 0 0 1 2 2 3 1 3 3 3 0 0 0	2 0 1 1 2 1 1 0 1 0 1 0

Problem J: Bacteria



In order to simplify the study of bacteria, their configuration space is sometimes downsized by modeling the microscopic pictures as a simple two-dimensional grid. As you can see a grid-simplification of a picture of a famous bacterium known as Dilbacterium, some cells of the grid are occupied by the body of bacteria, while the other cells remain empty.

Some empty cells of the grid are already occupied with other dead material. Such grid cells cannot be occupied by the bacteria. Let's G be the set of all cells of the grid that are not filled with dead material.

Two cells of a grid are called adjacent if they share a common edge. A sequence L of grid cells is called a path if every two consecutive cells of L are adjacent. A set S of grid cells is called connected if for all $c_1, c_2 \in S$, there is a path including c_1 and c_2 that only uses the members of S. Now, a grid-simplified bacterium (GSB) can be formally defined as a fixed-size connected subset of G.

As real living bacteria move, their movement should also be defined in the grid-simplified model. The movement of bacteria in this model has again a discrete definition. A single move of a GSB with the set of grid cells S, is defined as removing a grid cell $c_1 \in S$ and adding a grid cell $c_2 \in G - S$ such that the sets $S - \{c_1\}$, $S \cup \{c_2\}$ and $(S - \{c_1\}) \cup \{c_2\}$ are connected. Now, the set $(S - \{c_1\}) \cup \{c_2\}$ is the new valid configuration of the GSB after this single movement. By configuration, we mean the position of a GSB together with its shape.

A GSB can perform a series of movements in order to achieve a totally different configuration. Clearly, there are many different series of movements for a GSB with a single initial configuration, that all lead to the same final configuration. Given an initial and a target configuration of a GSB, your job is to find the minimum number of movements needed to lead its initial configuration to the target configuration. You can assume that the initial and the target configurations have no grid cells in common.

Input (Standard Input)

There are multiple test cases in the input. Each test case starts with a line containing space-separated integers m and n ($1 \leq m, n \leq 200$), which denote the height and width of the grid respectively. There are n characters in each of the next m lines which describe the structure of the $m \times n$ grid, and the initial and target configuration of the GSB. The characters of the grid can be one of the following:

- '`.`' representing empty cells
- '`#`' representing grid cells occupied with dead material
- '`x`' representing grid cells of the initial configuration
- '`o`' representing grid cells of the target configuration

It is guaranteed that the number of grid cells occupied in the initial and target configurations are equal and less than 10.

The input terminates with a line of the form "`0 0`" which should not be processed as a test case.

Output (Standard Output)

Write the result of the i^{th} test case on the i^{th} line of output.

For each test case, write the minimum number moves needed to lead the GSB from its initial configuration to the target configuration. If there is no way to reach the target configuration, write "`No solution`" (without quotation marks).

Sample Input and Output

Standard Input	Standard Output
2 3 ... x#o 2 3 x.o x#o 2 3 #.. x#o 9 9 .xxx..... .xxx...#. ########. ....#. .#####.#. .#.oo#.#. .#oooo.#. .#######. 0 0	4 3 No solution 41

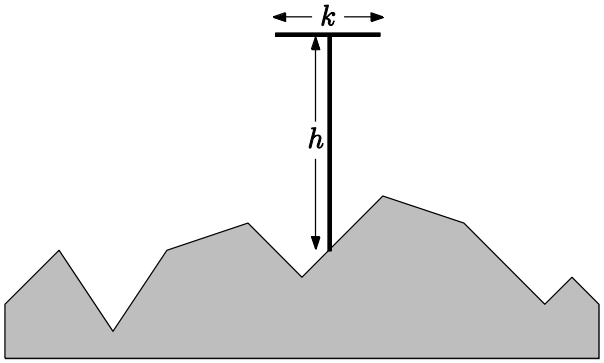
Problem K: T-cell Operator

T-cell operator which is famous for its T-shaped antennas is invited to launch a cellular network in a mountain range of Flatland, a well-known two-dimensional world. The mountain range can be modeled with a horizontal base line ($y = 0$) and an x-monotone polygonal terrain above the base line as depicted in the figure below. By x-monotone, we mean that the x-coordinate of the polygon vertices are in ascending order along the base line.

In the first step, T-cell gave the responsibility of overall planning and budget estimation to its operations committee (OC). The preliminary investigation of OC showed that developing several base stations (fixed-location transceivers also known as cell sites) is not economical as even a small piece of land is too expensive in the mountain range of Flatland. Fortunately, the mountain range is small enough to be totally covered by only one antenna, but the main problem is that T-cell uses signals which are unable to pass through solid objects (specifically the mountains here), i.e. one can communicate with the base station only if she can directly see at least a point of the antenna installed at the base-station.

As T-cell has got its world recognition due to the lack of any blind spot in its coverage, the final plan of OC is to install a T-shaped antenna (symmetric to its mast) with adjustable mast height h and the fixed head size k such that any point of the mountain range can see at least a point of the antenna. The antenna of course should not interiorly intersect the terrain. Touching the mountains is permitted though.

To make the plan economical, OC is now looking for a spot of the mountain range as the base station that minimizes h (the height of the antenna, where every point of the terrain can see at least one point of the antenna). And, that's why they have hired you; given the shape of the mountain range and the length k as the fixed head size of the antenna, find the optimum position of the base station with the minimum possible height h for its antenna.



Input (Standard Input)

The input contains several test cases. Each test case starts with a line containing two space-separated integers n ($2 \le n \le 1000$), the number of the vertices of the polygonal terrain, and k ($0 \le k \le 10^5$), the head size of the antenna. Next, the description of the terrain comes in n lines, one line per vertex. More precisely, the i^{th} line of this block contains two space-separated integers x_i and y_i ($0 \le x_i, y_i \le 10^5$), the coordinates of the i^{th} vertex (from the left) of the polygonal terrain, and indeed, the border of the mountain range can be obtained by connecting these consecutive vertices. It is guaranteed that $x_1 < x_2 < \cdots < x_n$, and the first and the last vertices of the terrain are not a peak (i.e. $y_1 < y_2$ and $y_{n-1} > y_n$). You can also assume that the line passing through each edge of the polygon does not intersect the line segment connecting the points $(0, 10^8)$ and $(10^5, 10^8)$.

The input terminates with a line of the form '`0 0`' (omit the quotes) which should not be considered as a test case.

Output (Standard Output)

For each test case, write a line containing the real number h, the minimum possible height of antenna for that position, rounded to exactly 2 digits after the decimal point.

Sample Input and Output

Standard Input	Standard Output
5 2 1 1 2 2 3 1 4 2 5 1 4 1 1 0 2 2 5 2 6 1 5 4 1 1 2 3 4 1 5 2 6 0 0 0	1.00 1.33 0.50