Problem A: Movie

The "Last ACM Contest" movie has been recently released and got the record of the highest worldwide gross. Gabby has been so busy due to his practicing for the ACM programming contest in Tehran site and hence, he has not succeeded to watch the movie at a cinema. Like many other people, he has legally downloaded the movie to watch it using a smart phone or a tablet on the way back home from the Sharif University on 20 December 2013, the contest day. Unfortunately, he has neither a smart phone nor a tablet. He therefore has decided to buy a device from the following smart phones or tablets that are available in the market.

Device	Price	Resolution
iPad Mini	319	1024×768
Galaxy Tab	419	1024×600
iPhone 4S	450	960×640
iPad4	519	2048×1536
iPhone 5C	599	1136×640
Galaxy Tab2	600	1280×800
Galaxy S4	630	1080×1920
iPhone 5S	719	1136×640

As you can see, each device has a known "height × width" resolution which is the number of distinct pixels in each dimension that can be displayed. Like the above devices, each movie has a known resolution $H \times W$ and video players equally scale both dimensions of the movie resolution by a factor of c to display it in a $\lfloor cH \rfloor \times \lfloor cW \rfloor$ window where $\lfloor x \rfloor$ is the largest integer not greater than x and c is a rational number such that cH and cW are not greater than the height and width resolutions of the display, respectively. Indeed, video players zoom a movie in or out while preserving the aspect ratio (the ratio of the height resolution to the width resolution) of the movie. When a video player displays a movie in the full-screen window in a device, some parts of the device display may remain "blank" (or unused) due to the difference in the aspect ratios of the device display and the movie. For a movie resolution and a display resolution, the usage ratio is defined to be the ratio of the non-blank area of the full-screen video-player window showing the movie to the area of the device display. As all above devices can rotate the whole screen 90 degrees, they may increase their usage ratio for a movie. For instance, if the movie resolution is 720×480, the usage ratio of iPad4 is the maximum of $(2048 * 1365)/(2048 * 1536)$ and $(1536 * 1024)/(1536 * 2048)$ (the latter is due to a 90-degree rotation) which is $1365/1536$.

Gabby is now looking for a device with the highest usage ratio for the "Last ACM Contest" movie which is stored with the resolution $H \times W$. He kindly asks you to report him this device before the end of the contest.

Input (Standard Input)

There are multiple test cases in the input. Each test case consists of a line containing two integers $1 \leq H, W \leq 5000$ which is the resolution of the "Last ACM Contest" movie. The input terminates with "0 0" which should not be processed.

Output (Standard Output)

For each test case, output a line containing the price of the device which has the highest usage ratio for the given movie resolution. In case of a tie, output the smallest price. For example, if both iPad4 and iPad Mini have the highest usage ratio, output 319.

Sample Input and Output

Standard Input	Standard Output
720 480 640 320 800 600 2500 2500 0 0	450 630 319 319

Problem B: SMS Poll

The popular TV show, "90%", is conducting a live SMS poll every week. Each poll simply consists of a question followed by k choices, numbered form 1 to k. The audiences are asked to vote to the choices by sending a number from 1 to k to the program's phone number via SMS. The poll statistics is shown instantly during the program.

The Parliament has recently got suspicious over the statistics broadcasted by this TV show last week, and has hired a committee to investigate the issue. The committee has obtained the list of all SMSs sent to the last program of 90%, and is going to process the data for investigation. The poll under investigation has only **four choices**.

Since the number of SMSs is pretty high, the committee is asking you to write a computer program to extract the poll statistics from the SMS raw data. The data is provided to you as a list of phone:content pairs, where phone is the sender's phone number, and content is the SMS content, presumably containing the sender's vote. Your task is to compute the percentage of votes for each of the choices, and report it to the committee. There are some points to be considered in processing data:

- In its most general form, a phone number is of the form "A B C", where A is the country code, B is the area code, and C is the local number. For example, in +98 (21) 6616-6645, thecountry code is 98, the area code is 21, and the local number is 6616-6645. None of A, B, and C can start with a 0 digit.
- The country code is optional, and is always preceded by a + sign. If the country code is not given, it is considered as 98. The country code consists of at most 3 digits.
- If the country code is given, the number must also contain an area code. However, if the country code is omitted, the area code is optional. But if the area code is provided, it must be preceded by either a 0 digit or it is surrounded by a pair of parentheses (for example, either 02166166600 or (21)66166600). If the area code is not given, it must be considered as 21. The area code consists of at most 3 digits.
- Local numbers can have various lengths, from 3 to 8 digits. Examples are 6616, and 66166600.
- A phone number may have been recorded in various formats. For example, 09128122190, +98(912)812-2190, and 0912-812-2190 all refer to the same phone number. Here are the format rules:
    - The area code can be optionally surrounded by a pair of parentheses.
    - There might be a dash (-) between any two digits in the local number or between two parts of the three parts namely the country code, the area code, and the local number. A dash and a parenthesis can't be adjacent.
- You can assume that all phone numbers in the raw data are valid and strictly follow the rules specified above. Moreover, you can assume that any phone number in the world has a unique complete form. For example, we cannot simultaneously have two different numbers (218)4460 and (21)84460, as they both have the same complete form +982184460.
- The only valid content for an SMS is a number from 1 to 4 (not surrounded by white spaces). Any other content, such as letters and + sign, makes the SMS invalid, and such SMSs must be discarded.
- A sender might have sent several SMSs from a single phone number. In this case, only the first valid SMS must be considered, and all others must be discarded.
- Discarded SMSs must not be included in the total number and in the percentages.

Input (Standard Input)

There are multiple test cases in the input. The first line of each test case contains a positive integers n ($1 \leq n \leq 10{,}000$), which indicates the number of SMSs in the list. The next n lines, each contains a pair $a{:}\ b$, where a is a phone number, and b is the SMS content. The content of each SMS is at most 30 characters where each character is an alphanumeric character or belongs to the set {"**:**", "+", "−", "(", ")"}. The input terminates with a line containing a single "0" which should not be processed. To make your life easier, it is guaranteed that there is no space character in the input.

Output (Standard Output)

For each test case, output four lines on the standard output, where line i contains the percentage of votes given to the i^{th}choice. The percentages must be **truncated** to integers. Then, output the total number of participants in the poll,

discarding duplicates SMSs. The format of the output must conform to the format indicated in the Standard Output below.

Sample Input and Output

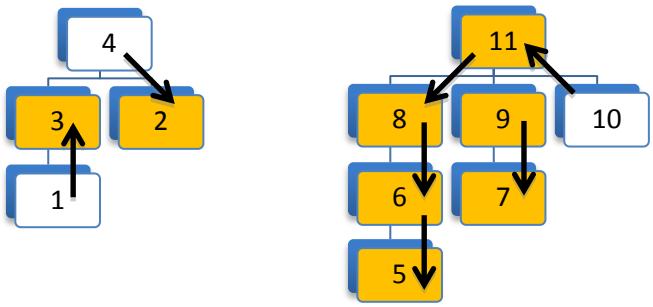
Standard Input	Standard Output
6 09128122190:1 +98211100:+1 +98(912)812-2190:4 +311(20)590-4359:2 6616:1 02166-16-00-22:2 8 6616:1 0216616:2 +98216616:3 +98(21)66-16:4x 9090:1 8080:2: +1(519)708-2040:3 (519)708-2040:3:1 0	50% 50% 0% 0% Participants:5 66% 0% 33% 0% Participants:5

Problem C: Traitor

According to an intelligence source, we've got a traitor in ACM Security Agency (ASA). ASA has a hierarchical structure where each agent has a manager and there are also some (at least one) top managers who are not managed by anyone. Our source doesn't exactly know the traitor, but he has a list of suspects. Therefore, all we know is that there is exactly one traitor in our agency and we have a list of suspects. In order to find that traitor, we want to assign a watcher for each suspect, satisfying the following three conditions:

1. Two suspects cannot watch each other.
2. Each suspect should be watched by either his manager or one of his direct employees.
3. Nobody can watch more than one suspect.

If we want to satisfy all above conditions, it may be impossible to watch all suspects. Therefore, you should write a program that gets the structure of ASA and the list of suspects as the input and finds the maximum number of suspects for whom the watcher assignment is possible. In the following figure that illustrates the organizational structure of ASA with two top managers and eleven agents, the suspects are indicated with gray color. One watcher assignment covering 7 out of 8 suspects is possible in this case which is shown by arrows. An arrow from agent *x* to agent *y*, means agent *x* is supposed to watch agent *y*. It can be shown that in this example there is no watcher assignment covering all suspects.



Input (Standard Input)

There are multiple test cases in the input. The first line of each test case contains two integers n ($1 \leq n \leq 10{,}000$) specifying the number of agents, and k ($1 \leq k \leq n$) specifying the number of the suspected agents. The agents are numbered from 1 to n. On the second line there are n space-separated integers, where the i^{th} number is the number of agent who is the manger of the agent i. A zero means the agent i is a top manager. On the third line there are k positive integers $s_1, s_2, \ldots, s_k$, indicating the numbers of the suspected agents. The input terminates with "0 0" which should not be processed.

Output (Standard Output)

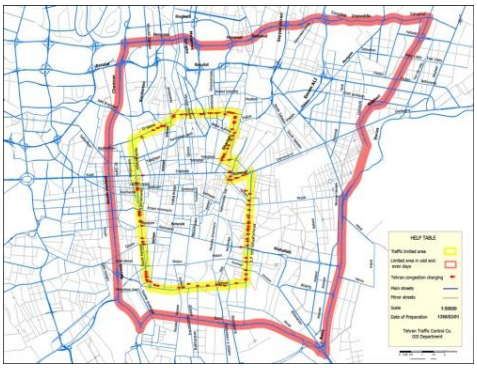
For each test case, output in a line the maximum number of suspects for whom the watcher assignment is possible.

Sample Input and Output

Standard Input	Standard Output
11 8 3 4 4 0 6 8 9 11 11 11 0 3 2 8 9 6 5 7 11 2 2 0 1 1 2 0 0	7 1

Problem D: Intelligent Traffic Surveillance

Recently, Mr. Hamsadeh is working as an IT engineer in the central traffic control department of Tehran municipality. He is now in charge of a new traffic surveillance system with which the law-breaking vehicles are going to be fined through automatic plate detection and issuing penalty tickets. The system has been running for a few weeks in the production environment and now is the time for Mr. Hamsadeh to issue the tickets. While he was preparing the system, the persistence storage of the database server came into a fatal hardware crash and all its data was disastrously lost. Laughing at his own misery, misfortunate Mr. Hamsadeh started inspecting the other systems in hope of data-recovery. The only available pieces of information are the log files remaining in the application server. Fortunately, all forms of data-entry are logged in the service layer of the application server, and thus, the lost information can be totally survived using the log files. But such data-recovery tasks are too complicated for Mr. Hamsadeh and there is not enough time to setup a new database, import the recovered data and reconfigure the whole architecture for issuing the tickets. With deep feelings of being squashed, Mr. Hamsadeh is asking you for help. You have to write a program that reads all the log files and issues the penalty tickets. In the following paragraphs, Mr. Hamsadeh describes Tehran traffic regulations, services provided by the traffic surveillance system, the format of log files, and the rules for issuing tickets.



Each road is considered in one of the three traffic-specific zones of Tehran:

- Central Traffic Restricted Zone (CTRZ)
- Even/Odd Restricted Zone (EORZ)
- Unrestricted Zone (UZ)

Ordinary personal vehicles are not normally permitted to enter CTRZ during these time intervals:

- Saturday through Wednesday from 06:30 to 17:00
- Thursday from 06:00 to 13:30

Ordinary personal vehicles whose vehicle registration number ends with an even digit are not permitted to enter EORZ during these time intervals:

- Sunday and Tuesday from 06:30 to 19:00
- Thursday from 06:30 to 17:00

Ordinary personal vehicles whose vehicle registration number ends with an odd digit are not permitted to enter EORZ during these time intervals:

- Saturday, Monday and Wednesday from 06:30 to 19:00

So, none of these zone restrictions are applied on Fridays (official weekends in Iran). Some vehicles (including public transportation and emergency services) are allowed to enter CTRZ and EORZ anytime with no restrictions. Ordinary vehicles can also enter these zones if they buy the permission for a single day, but you are not to consider buying permissions in this problem. If a vehicle enters CTRZ or EORZ unlawfully, it should be fined. Each vehicle must be fined at most once a day for zone restriction violations. If both CTRZ and EORZ violations happen for a vehicle on a single day, the CTRZ violation is considered which has naturally a higher penalty. Each road is initially considered to be in UZ, but this state may change based on announcements. The new rules of all such announcements must be applied from the day after the announcement. In the same way, no vehicles are initially exempt from zone regulations, but the exceptions are added and removed over time. Such modifications must also be considered from the next day.

The following are the services provided by the application server. Each service has a corresponding log message which is written on a single line starting with the service name followed by the parameters. As you will see in service definitions and examples, the parameters of a service are printed in a specific order. Independent of the service type, the parameter list of each log message starts with a special pair of parameters (called *timestamp*): "`day`" and "`time`". A timestamp naturally shows the exact time of its corresponding service request. Parameter "`day`" is a positive integer showing the number of days passed from the day of system deployment (called day 0). Parameter "`time`" shows the time of the request on that day in "$HH:mm:ss$" format ($00 \leq HH \leq 23$, $00 \leq mm, ss \leq 59$).

- setRoadZone(day, time, zone, roads)
  Parameter "`zone`" can be "`UZ`", "`CTRZ`", or "`EORZ`". Parameter "`roads`" contains a non-empty list of not-necessarily-distinct road names. From the beginning of the next day, the corresponding roads must be considered in the given "zone" (overriding their former zone states). Log examples:
  - `setRoadZone 2 "09:12:53" "CTRZ" "Enghelab" "Sa'di" "Ferdowsi" "Ferdowsi"`
  - `setRoadZone 5 "14:32:01" "EORZ" "Resalat"`
  - `setRoadZone 12 "00:00:59" "UZ" "Persian_Gulf"`
- addZoneException(day, time, vehicles) | removeZoneException(day, time, vehicles)

Parameter "vehicles" contains a non-empty list of not-necessarily-distinct vehicle registration numbers. From the beginning of the next day, the corresponding vehicles {must not | must} be fined in the case of entering CTRZ or EORZ during the forbidden times. These commands override the older state of the given vehicles (which might be the same state). Log examples:

- `addZoneException 1 "09:00:13" "1234567" "9876543"`
- `removeZoneException 3 "15:33:02" "1234567" "9876345"`

- addPhotoInfo(day, time, photoId, road, vehicles)
  This is one of the key services of the system. It is not called from user interface layer. It is called by another complex system: the external image processing server which analyzes the pictures taken by the surveillance cameras. This service is called when a photo is taken and analyzed. The identifier of the analyzed photo is given in parameter "photoId", a positive integer. Parameter "road" holds the name of the road from which the photo was taken. The image processing server detects the vehicles in the photo and extracts their registration number from their plates. Parameter "vehicles" provides the list of these vehicle registration numbers. It is possible for this list to be empty as there might be no vehicles in the photo. The "timestamp" parameter here refers to the moment of taking the photo. Log examples:
  - `addPhotoInfo 18 "03:18:43" 3324249 "Pastor" "6256256" "8888310"`
  - `addPhotoInfo 4 "20:47:31" 54 "Mokhberoddoleh,_sar-e_Sa'di"`
  - `addPhotoInfo 27 "06:39:14" 112385612 "17-e_shahrivar" "1006016"`

You can assume the following statements:

- Parameter objects are always in one of the following forms:
  - Integer numbers: All nonnegative
  - Strings: Always surrounded with quotation marks (")
  - Lists: Always appearing as the last parameter, consisting of pace-separated strings through the end of the line
- All tokens including service names and parameter objects (strings, integers, and list members) are separated with a single space.
- The "time" parameter which is in "$HH:mm:ss$" format, has exactly 8 characters and all its three parts are exactly 2 digits (padded with "0" in the case of being less than 10).
- Road names consist of English alphabet (both lowercase and uppercase letters), digits, dash (-), underscore (_), dot (.), comma (,), and single quotation mark ('). Road names are nonempty and no longer than 100 characters.
- All vehicle registration numbers are strings of exactly 7 digits.
- Each road or vehicle is always referenced with the same string.
- No two timestamps are exactly the same.
- If there are conflicting commands on the same day, the newer command (one with the bigger timestamp) must override the older one. You can assume that the "addPhotoInfo" service is called at most once for each photoId.
- Each vehicle registration number appears at most once in each photo.
- The running time of the system is at most 300 days.
- A vehicle must be fined for outlawed zone entrance even if its photo was taken on the time boundaries (e.g. at Monday 6:30:00).

The vehicles must be fined based on photo analysis results. As stated before, zone-entrance penalty tickets should be issued for each vehicle at most once a day. All photos of such a violation in a day must be attached to its corresponding penalty ticket. Photo attachments of a ticket must be sorted in ascending order based on their times. Note that if both CTRZ and EORZ entrances happen together for a vehicle, all photos of both violations must be attached, but the ticket should be issued with the CTRZ penalty. The order of printed tickets is also important. Tickets must be primarily sorted based on their vehicles such that their registration numbers appear in lexicographic order. Tickets of a vehicle must then be sorted in ascending order based on the day of offence.

Input (Standard Input)

The input consists of several test cases. Each test case starts with a line containing the single integer N, the total number of lines in the log files ($1 \leq N \leq 1000$). The second line contains a string W followed by two positive integers CTP and EOP. String W is one of the words "Saturday", "Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", or "Friday", specifying the weekday of the deployment day (day 0). Numbers CTP and EOP are the penalty values for outlawed entrance to CTRZ and EORZ respectively ($EOP < CTP$). Each of the next N lines contains a service log in the format specified before. The logs are not necessarily sorted in any special order as it is the concatenation result of many log files. Each line has at most 1000 characters. All integers in the input are less than 10^9. The input terminates with a line containing "0" (omit the quotes).

Output (Standard Output)

For each test case, you have to print the penalty tickets in the formerly-specified order. Each ticket has a main body and one or more attached photos. The main body must be printed first in a single line with the following format:

`vehicle: "`*NUMBER*`", day:` *DAY*`, offence: "`*OFFENCE*`", penalty:` *PENALTY*

Here is the meaning of the placeholders in this template:

- *NUMBER*: The registration number of the law breaking vehicle
- *DAY*: The day of offence; numbered like timestamp parameters
- *OFFENCE*: It specifies the type of offence and is always one of the following strings:
  - `Outlawed entrance to CTRZ`
  - `Outlawed entrance to EORZ`
  - `Outlawed entrance to CTRZ & EORZ`
- *PENALTY*: The price that the vehicle driver is charged in this ticket

Each photo attachment must appear in a separate line. They must be printed in the order specified before. Photo attachments must have the format below:

`photo:` *PHOTO-ID*`, time: "`*TIME*`", road: "`*ROAD*`"

Here is the meaning of the placeholders in the template of photo attachments:

- *PHOTO-ID*: It is the identifier of the attached photo. It is used by the real ticket renderer to print the photo.
- *TIME*: The time of taking the corresponding photo in "*HH*:*mm*:*ss*" format (exactly the same as timestamp)
- *ROAD*: Name of the road on which the corresponding photo was taken

Follow the specified formats precisely, especially the order of parameters, spacing, punctuations, and quotation marks. Print a line containing "`###`" between every two consecutive test cases.

Sample Input and Output

Standard Input

```
6
Friday 30000 25000
setRoadZone 1 "09:12:53" "CTRZ" "Enghelab" "Ferdowsi" "Behesht"
setRoadZone 1 "14:32:01" "EORZ" "Resalat" "Damavand"
addZoneException 1 "09:00:13" "1000100" "1000200" "1000300"
addPhotoInfo 2 "13:18:43" 1004 "Enghelab" "1000100" "1000200" "1000400" "1000105"
addPhotoInfo 3 "11:55:12" 1003 "Behesht" "1000400" "1000105"
addPhotoInfo 2 "06:30:00" 1002 "Resalat" "1000100" "1000105" "1000120" "1000400"
3
Sunday 100 90
setRoadZone 4 "01:02:03" "CTRZ" "16-e_Azar"
addPhotoInfo 4 "10:15:13" 211 "16-e_Azar" "1010101"
addPhotoInfo 5 "20:21:42" 212 "16-e_Azar" "2020202"
6
Monday 1000 900
setRoadZone 13 "09:00:00" "CTRZ" "Azadi"
addPhotoInfo 13 "10:00:00" 101 "Azadi" "1000001"
addPhotoInfo 14 "10:00:00" 102 "Azadi" "1000001"
setRoadZone 15 "09:00:00" "UZ" "Azadi"
addPhotoInfo 15 "10:00:00" 103 "Azadi" "1000001"
addPhotoInfo 16 "10:00:00" 104 "Azadi" "1000001"
0
```

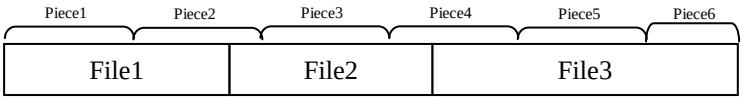
Standard Output

```
vehicle: "1000105", day: 2, offence: "Outlawed entrance to CTRZ", penalty: 30000
photo: 1004, time: "13:18:43", road: "Enghelab"
vehicle: "1000105", day: 3, offence: "Outlawed entrance to CTRZ", penalty: 30000
photo: 1003, time: "11:55:12", road: "Behesht"
vehicle: "1000120", day: 2, offence: "Outlawed entrance to EORZ", penalty: 25000
photo: 1002, time: "06:30:00", road: "Resalat"
vehicle: "1000400", day: 2, offence: "Outlawed entrance to CTRZ & EORZ", penalty: 30000
photo: 1002, time: "06:30:00", road: "Resalat"
photo: 1004, time: "13:18:43", road: "Enghelab"
vehicle: "1000400", day: 3, offence: "Outlawed entrance to CTRZ", penalty: 30000
photo: 1003, time: "11:55:12", road: "Behesht"
###
###
vehicle: "1000001", day: 14, offence: "Outlawed entrance to CTRZ", penalty: 1000
photo: 102, time: "10:00:00", road: "Azadi"
vehicle: "1000001", day: 15, offence: "Outlawed entrance to CTRZ", penalty: 1000
photo: 103, time: "10:00:00", road: "Azadi"
```

Problem E: BitTorrent

The BitTorrent is a protocol for transferring large files running over a peer-to-peer network in which nodes act as both clients and servers, in contrast to the centralized client–server architecture where client nodes request central servers to get resources. Indeed, the protocol allows users to establish a group of hosts to download and upload files from each other simultaneously. Precisely, the whole package of files (so-called torrent) is segmented into *pieces* as depicted in the figure. For instance, a 10 MB package might be segmented into exactly ten 1M-size pieces or exactly forty 256KB-size pieces. As each host (or peer) receives a new piece of the torrent it becomes a source of that piece for other hosts willing to have that piece. Pieces are typically downloaded non-sequentially and are rearranged into the correct order by hosts. Each host independently manages which pieces must be downloaded. Pieces are of the same size throughout a single torrent download except the last piece which may have a smaller size.

You want to download a package of files, but you are approaching your monthly Internet usage limit and you don't want to wait till the next month. You want to download the maximum number of files with the bandwidth left. Which pieces must be downloaded?



The package of files (torrent)

Input (Standard Input)

The input contains multiple test cases. Each test case starts with three space-separated integers N, P, and L where N is the number of files in the torrent ($1 \leq N \leq 3{,}000$), P is the size of pieces in KB ($1 \leq P \leq 1000$), and L is the remaining kilobytes from your monthly Internet usage limit ($1 \leq L \leq 10^6$). The second line of a test case contains N space-separated positive integers not exceeding $100{,}000$ where the i^{th} integer is the size (in KB) of the i^{th} file in the torrent. The input terminates with "0 0 0" which should not be processed.

Output (Standard Output)

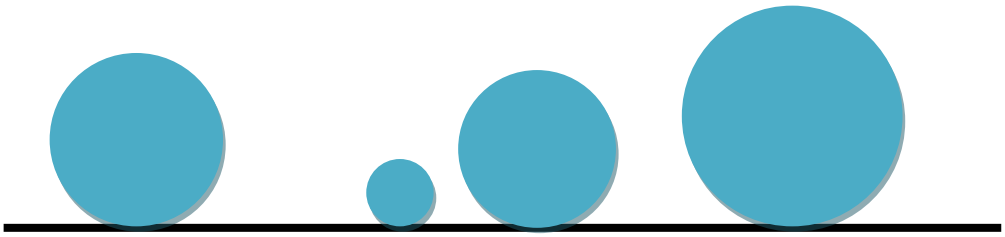
For each test case, output a line containing the maximum number of files which can be downloaded from the torrent.

Sample Input and Output

Standard Input	Standard Output
3 3 13 5 5 7 7 2 16 6 11 3 3 8 1 8 0 0 0	2 4

Problem F: 2D-Solar System

The 2D-solar system like our solar system comprises Bigsun (its sun) and its planetary system of many circular planets orbiting around Bigsun. Due to the high gravity of Bigsun, all planets have been attracted by Bigsun. Precisely, they orbit around Bigsun while being tangent to it as depicted in the figure (As Bigsun is so huge, its boundary looks like a line.) Surprisingly, up to the current time no two planets have collided with each other, but no one knows whether the system is free of collisions in the future. You are to write a program to verify whether there is a possibility of any collision in the future and if so, compute the time at which the first collision happens. The scientists of NASA have realized that each planet in the 2D-solar system moves with a constant velocity. More precisely, it turned out that the motion equation of a planet can be described by the position of its touching point with the boundary of Bigsun through time by the linear equation $y = at + b$ where a and b are two known parameters and t denotes time.



Input (Standard Input)

There are multiple test cases in the input. Each test case starts with a line containing an integer n ($0 \leq n \leq 50{,}000$) where n is the number of planets. The i^{th} line of the next n lines contains 3 space-separated integers r_i, a_i, and b_i whose absolute values are not exceeding 1,000,000,000. The number r_i which is a positive square number, denotes the radius of Planet i and a_i and b_i specify its motion equation, i.e. the position of the tangent point of the planet on the boundary of Bigsun at time t is $a_i t + b_i$. The input terminates with a line containing "0" which should not be processed.

Output (Standard Output)

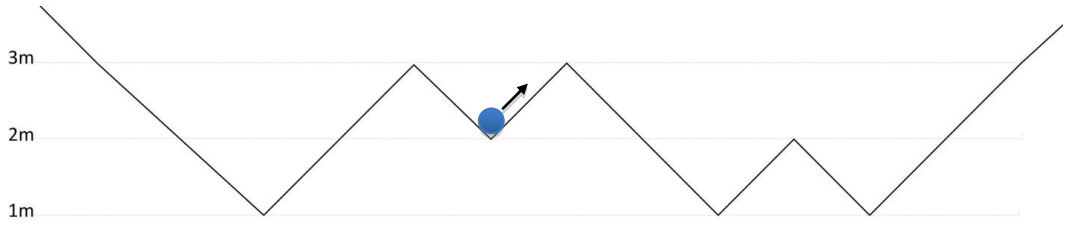
For each test case, output a line containing the time at which the first collision happens under the assumption that the current time is equal to 0 and all planets are disjoint at the current time. If the system is free of collisions you must output "Collision-Free System". The output must be rounded to exactly two digits after the decimal point.

Sample Input and Output

Standard Input	Standard Output
3 1 1 1 4 3 6 9 -7 30 2 4 -1 1 1 1 7 2 1 1 10 1 2 5 0	1.20 Collision-Free System 3.00

Problem G: Bowling Ball

There are a lot of mountain ranges in the Neverlands. Every mountain range consists of a number of valleys and summits. The slope between two consecutive summit and valley is always either 1 or -1, and all the summits and the valleys have integer heights. A bowling ball is swinging on a part of a mountain range consisting of n valleys and $n-1$ summits. The ball is always touching down the surface of the mountain range (it does not jump). Mountains before the first and after the last valleys are too high such that the ball can never exit the mountain range. At time t_0 the ball is located in the valley number s moving to the upper-right direction and it has an initial kinetic energy K_0. The following figure shows a mountain range with 4 valleys and 3 summits, and the ball located in the 2nd valley (enumerated from left to right).



By the simple physics we know that at any time t the ball has a gravitational potential energy $P_t = mgh$ and also a kinetic energy $K_t = (\frac{1}{2})mv^2$, where m is the mass of the ball, g is the constant of the Earth gravity (here equals to 10), and h and v are the height and the velocity of the ball at time t. By the transformation of energy from potential to kinetic or vice versa, the total energy of the ball $P_t + K_t$ is fixed during its movements, unless it falls into a valley: at the i^{th} valley (from left), c_i units of the kinetic energy is lost due to friction or it stops if its kinetic energy is below c_i but consider no friction in other locations. Note that the ball loses c_s unit of energy when it leaves the starting valley at time t_0. You can assume the ball diameter is equal to 0 and its mass is equal to 1. Your task is to find the valley or summit at which the ball will stop.

Input (Standard Input)

There are multiple test cases in the input. The first line of each test case contains three space-separated positive integers n, and s and K_0 ($1 \le n \le 3000,\ 1 \le s \le n, 1 \le K_0 \le 10^{15}$). Each of the following n lines contains two integers h_i and c_i, the height and friction of the i^{th} valley. The j^{th} line of the next $n-1$ lines contains H_j, the height of The j^{th} summit from the left ($0 \le h_i, c_i, H_j \le 10^9$). It is guaranteed that at least one of c_is is greater than zero. The input terminates with "0 0 0 0" which should not be processed.

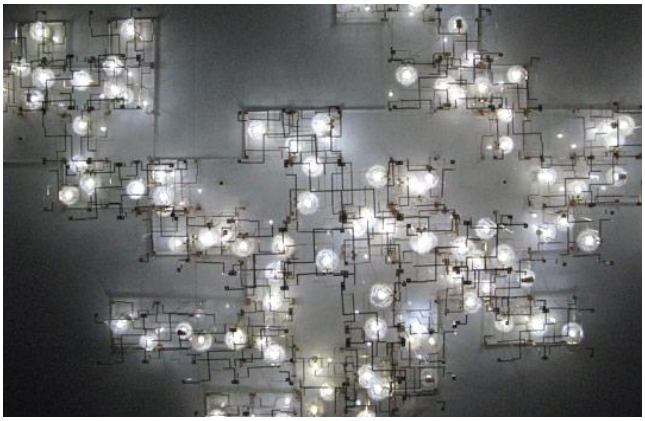
Output (Standard Output)

For each test case, output a line conforming one of the following formats depending on whether the ball stops at either a valley or a summit.

- If the ball stops at valley number k, output "Valley: k" (omit the quotes.)
- If the ball stops at Summit number k, output "Summit: k (omit the quotes.)

Sample Input and Output

Standard Input	Standard Output
4 2 17 1 1 2 1 1 1 1 2 3 3 2 1 1 1000000000000000 1 1 0 0 0 0	Summit: 2 Valley: 1



Problem H: LED Circuit

You are in charge of preparing the conference hall for the closing ceremony of ACM ICPC. You had hired a perfectionist sentimental room designer to design an elegant decoration for the old hall. The ceremony is going to start in a few hours and the designer has just informed you of the completion of the decoration. When you reach the conference hall, you confront a strange circuit on the wall. The designer starts explaining the meanings of life and ACM ICPC hidden under each piece of the circuit. The circuit consists of LEDs (Light Emitting Diodes) interconnected with junctions and wires. You ask whether the LEDs should be switched on. "Of course!" the designer responds, "Each and every LED must be lit, otherwise the whole work is junk!" You look around the circuit to find its power plug.

- Where is the power plug?
- Huh? It's beyond the scope of my work! It is you who has to provide the electricity to the LEDs. And be careful! Do not remove or modify a single part of the circuit; Just connect power supply cables to the junctions. I have to leave right now. I will send a report of my perfect work to your manager. Bye.

Left alone with the bizarre circuit, you start inspecting the circuit.

Unlike incandescent light bulbs, which emit light regardless of the electrical polarity, LEDs only emit light with the correct electrical polarity (i.e. when their anode pin has a higher electric potential than the cathode pin). Each LED has a minimum voltage. An LED does not emit (even with the correct polarity) if the electrical potential difference of its pins is less than its minimum voltage. Each LED has also a maximum voltage. An LED is destroyed if its electrical potential difference exceeds its maximum voltage.

Your inspection on the circuit shows it consists of three types of components:

- LEDs: Fortunately, all LEDs of the circuit are of the same type, and so having the same minimum voltage, and the same maximum voltage.
- Junctions: Each of the two pins of an LED in the circuit is connected to a junction. Junctions are not only a place for connecting the LED pins, but also for connecting the wire end-points.
- Wires: Each wire has two end-points and connects a junction directly to another junction forcing them to have the same electric potential.

By connecting external electrical poles (with different values of voltage) to each of the junctions of the circuit, you can inject different electric potentials to the junctions. Note that each junction must be connected to an external electrical pole. Be careful of *short circuits*: the end-points of each wire MUST have the same electric potential. By convention, we can assume the minimum electric potential to be zero. So, all the electric potentials can be considered to be nonnegative.

Now, you have to buy an external power supply that provides you with the required electrical poles. The cost of such power supplies depends on their *upper-bound*: the maximum voltage they provide.

Given the specification of the LED circuit, you have to write a program that tests if it is possible to light all the LEDs *correctly* (with no short circuits, and no LED destruction). In the case of the possibility, the program should also compute the minimum possible upper-bound of power supply with which all LEDs can emit light.

Input (Standard Input)

The input consists of several test cases. Each test case describes an LED circuit and starts with a line containing 5 space-separated integers J, L, W, m, and M, where J is the number of junctions ($2 \leq J \leq 500$), L is the number of LEDs ($1 \leq L \leq 5{,}000$), W is the number of wires ($0 \leq W \leq 5{,}000$), and m and M are the minimum and maximum voltage of LEDs respectively ($1 \leq m < M \leq 1000$). Assume that the junctions are numbered 1 through J.

Each of the next L lines represents an LED with two space-separated integers. The first integer is the number of the junction to which the anode pin of the LED is connected and the second integer is the number of the junction for the

cathode pin. Then, *W* lines follow, each describing a wire of the circuit using two space-separated integers: the junctions the wire directly connects.

The input terminates with a line containing "`0 0 0 0 0`" (omit the quotes).

Output (Standard Output)

Write a single line of output for each test case. If there is no way to correctly light all the LEDs in the circuit of that test case, only write the word "`Impossible`" (with no quotes). Otherwise, write a single integer: the minimum upper-bound of power supply with which all the LEDs can be lit.

Sample Input and Output

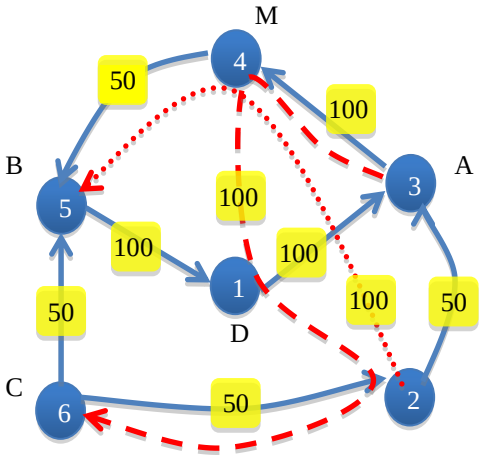
Standard Input	Standard Output
2 1 0 3 5 1 2 3 2 0 3 5 1 2 3 2 3 2 0 3 5 3 2 1 3 3 1 1 3 5 1 2 2 3 3 1 2 3 5 1 2 2 3 3 1 3 3 0 3 5 1 2 2 3 1 3 3 3 0 3 6 1 2 2 3 1 3 4 2 2 2 7 1 2 3 4 3 1 2 4 4 2 2 2 7 1 2 3 4 3 2 1 4 0 0 0 0 0	3 3 6 3 Impossible Impossible 6 2 Impossible

Problem I: Mixed Flight Plans

Ad Hoc Postal Company (AHPC) has a simple strategy to deliver postal envelopes: It advertises for a volunteer and buys for him/her the cheapest flight-plan from the source to the destination city, and gives the volunteer a bag of envelopes in the source airport to be handed over the company's correspondent in the destination airport.

In addition to direct flights between two airports, there are indirect trips with stops in several different intermediate airports. An indirect trip passenger must always start from the first airport and visit the intermediate airports consecutively (according to the indirect trip schedule, without using any other direct flights or indirect trips in the middle) but may leave the rest of the indirect trip at any intermediate airport. The price of an indirect trip is fixed; no matter if the passenger uses all the flights or interrupts in the middle. A flight-plan can consist of several direct flights and whole or partial of indirect trips.

In sake of economy, AHPC is looking to use mixed flight plans: suppose a couple of bags, one should be delivered from *A* to *B* and another from *C* to *D* (*A*, *B*, *C*, *D* are different airports). The company may buy two flight plans from A to D for the first volunteer, and from C to B for the second one, such that the two plans share the same airport M (not necessarily different from these 4 airports) at which the two volunteers meet and exchange their bags. Hence, AHPC ensures each bag is delivered to the right destination, while the total price might be reduced.



As an example, six airports together with direct flights (solid arrows), indirect trips (dash and dot arrows) and their prices are illustrated in the above figure. Two bags should be sent, one from the airport 3 (A) to 5 (B), another from 6 (C) to 1 (D). The company might purchase (3,4), (4,5) as the flight-plan of the first volunteer, and (6,5), (5,1) for the second with the total cost of 300. But the cheaper alternative would be (3,4,1,2,6) for the first and (6,2), (2,4,5) for the second volunteer, with the total cost of 250. The volunteers meet and exchange bags at airport 4 (M) and the first volunteer will leave the indirect trip at airport 1.

You should write a program that given flights information, finds the most economic cost for delivery of the bags.

Input (Standard Input)

There are multiple cases in the input. The first line of each case contains six integers n, m, A, B, C, and D, where n ($4 \leq n \leq 100$) is the number of airports, and m ($0 \leq m \leq 10{,}000$) is the total number of direct flights and indirect trips, where at most 1000 of them are indirect trips. The i^{th} line of the next m lines starts with two positive integers p_i (the price, $p_i \leq 10^6$) and s_i (being 1 for direct flight and more for indirect trip), followed by $s_i + 1$ distinct airport numbers that show the order of airports visited in the i^{th} direct flight/indirect trip. The input terminates with "0 0 0 0 0 0" which should not be processed.

Output (Standard Output)

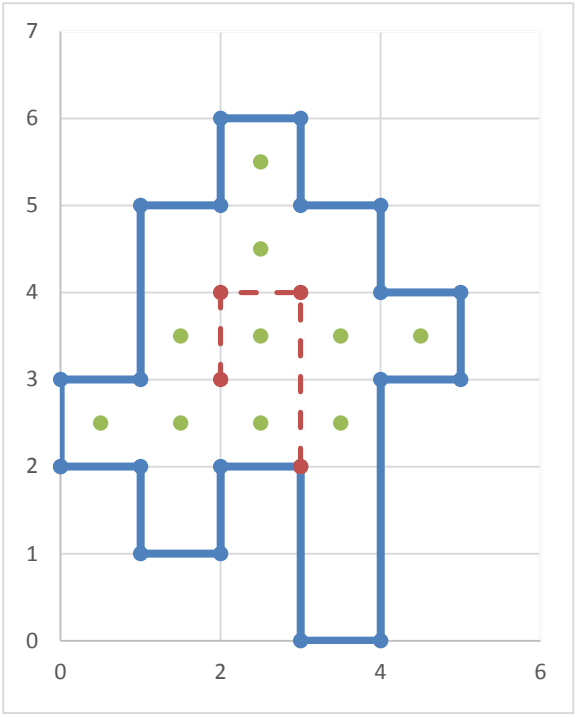
For each test case, output the total cost of the cheapest flight plan in a line, or output "Impossible!" (without quotes) if delivery of at least one bag is impossible.

Sample Input and Output

Standard Input	Standard Output
6 9 3 5 6 1 100 1 3 4 50 1 6 2 100 2 2 4 5 50 1 6 5 100 1 1 3 100 4 3 4 1 2 6 100 1 5 1 50 1 4 5 50 1 2 3 4 0 1 2 3 4 5 2 1 2 3 4 10 4 1 2 5 3 4 20 1 3 5 0 0 0 0 0 0	250 Impossible! Impossible!

Problem J: Sweeping Robot

A robot equipped with a camera is to sweep some areas in a museum. The museum is a polygon with only horizontal and vertical walls as depicted in the figure. We assume that the polygon is drawn on an underlying mesh of 1x1 cells and its vertices are on the mesh vertices. The robot is also restricted to move only along the mesh edges either horizontally or vertically. Its camera sweeps everything that can be seen on the perpendicular directions of its moving path. That is, when the robot moves horizontally, its camera is directed vertically and can only see any visible items located on the north and south of the horizontal segment of the moving path. Similarly, its camera can see any visible cells located on the east and west of the vertical segment of the moving path. In the figure, a polygon and a moving path of our robot (the dashed line path) are shown. Here, the dotted squares are seen.



Given such a polygon and a robot-path inside it, the problem is to compute the total areas (total number of squares) seen by the robot.

Input (Standard Input)

There are multiple test cases in the input. Each test case starts with a line containing two integers n and k ($2 \leq n, k \leq 100$) where n is the number of walls (or vertices) of the museum and k is the number of the vertices of the robot path. The next n lines describe the museum vertices. The i^{th} line contains 2 space-separated non-negative integers x_i and y_i not exceeding 500 denoting the x and y coordinates of the i^{th} vertex of the museum, respectively. There is a wall between vertices i and $i+1$ (You may assume the $(n+1)^{\text{th}}$ vertex is the first vertex). The next k lines describe the vertices of the robot path in order of appearing on the path from the starting point to the ending point; each line contains two integers which are the x and y coordinates of a vertex. The robot path is guaranteed to be inside the museum but its vertices (not its edges) can touch the museum walls. Note that the robot path may intersect itself. The input terminates with a line containing "0 0" which should not be processed.

Output (Standard Output)

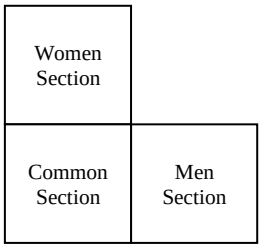
For each test case, output the total area (total number of squares) seen by the given robot.

Sample Input and Output

Standard Input	Standard Output
20 4 0 2 1 2 1 1 2 1 2 2 3 2 3 0 4 0 4 3 5 3 5 4 4 4 4 5 3 5 3 6 2 6 2 5 1 5 1 3 0 3 2 3 2 4 3 4 3 2 0 0	10

Problem K: Wedding Hall

Kamran has recently bought a rectangular flat garden in an awesome part of the countryside. His business plan is to construct a Hall to host wedding ceremonies, since the countryside has recently attracted a lot of attention for being a fantastic area to host wedding ceremonies. As women and men sections must be separated based on the nation law, he thinks of designing the hall in three sections: men section, women section and common section (including rest rooms, dinner room and etc). As the common section must be easily accessible to all persons, it must be designated in the middle of the other two sections. Among several proposal designs, Kamran has selected the one depicted below where all three sections are squares of the same size, they are attached to each other like an L shape, their sides are parallel to the garden sides, and the visible sides of the common section from outside face the south and west of the garden. Now the main question is where the hall must be constructed. The garden is full of old trees and cutting the trees is forbidden due to high air pollution. He kindly asks you to help him to find the largest hall that he can construct.



Input (Standard Input)

There are multiple test cases in the input. Each test case starts with a line containing a non-negative integers n ($1 \leq n \leq 50{,}000$) and two positive integers a and b (all not exceeding 1,000,000) where n is the number of distinct trees in the garden and a and b specify the sides of the garden. Precisely, $[0, a] \times [0, b]$ denotes the rectangle modeling the garden. The next n lines, each contains 2 space-separated non-negative integers x_i and y_i ($0 < x_i < a$, $0 < y_i < b$) denoting the x and y coordinates of a tree, respectively. You may assume that trees have distinct coordinates and the south side (i.e. $[0, a]$) and the west side (i.e. $[0, b]$) of the garden lie on the x-axis and the y-axis, respectively. The input terminates with a line containing "0 0 0" which should not be processed.

Output (Standard Output)

For each test case, output the area of the largest hall that Kamran can construct in his garden. Note that the hall can touch the trees or the garden sides but it can't interiorly include them. The output must be rounded to "exactly" two digits after the decimal point.

Sample Input and Output

Standard Input	Standard Output
2 3 5 2 2 1 4 0 0 0	6.75