ACM ICPC SOUTH PACIFIC REGIONAL FINALS

September 26, 2015

Contest Problems

Sponsored by:
WiseTech GLOBAL

This Page Is Intentionally (Almost) Blank.

A: Appropriate Coordinate Map

Time Limit: 5 second(s)

Your country is about to install a new hyperfast broadband network based on an advanced technology, which allows a single unidirectional link to carry any amount of data traffic at any speed. Each such link runs straight between its endpoints, similar to an ideal laser beam. Given the unidirectional nature of the connections and the astronomic expense of link endpoints, the network's nodes will be connected in a ring (a *ring* is a connected graph where each node has exactly one input and one output). This will minimise the expense given the constraint that each node in the network must be able to send data to, and receive data from, each other node (either directly or via intermediate nodes). There is an election looming and although there is bipartisan agreement on the list of potential locations, the government and opposition disagree on which of the potential locations should be nodes on the network.

The opposition, which favours fiscal rigour, is attempting to sound impressive while keeping the cost down:

> "Our network will include just enough nodes such that if these are then connected into a ring with minimal total link length, then every location will either be on the network, directly passed over by a link, or in the area enclosed by the network."

In contrast, the government, which favours equality of network access and does not hesitate to spend taxpayers' money, insists that all the potential locations will be part of the network. As the government's Chief Computationalist, you meet with the Communications Minister to hear the government's further requirements:

> "We need a connection scheme that we can explain to the public in a simple way. To achieve this, even though a ring doesn't really have ends, I want two nodes, say A and B, to be nominated as the ring endpoints and an 'Appropriate Coordinate Map' (ACM) set up with the origin at A and the positive $x-$axis oriented to pass through B, whose distance unit is the same as our usual one. The ring must then be constructed such that the path through the ring from A to B is in non-decreasing order of the ACM $x-$axis and the path from B to A is in non-increasing order of the ACM $x-$axis. This will enable me to explain A to B as a consistent outward leg, and B to A as a similarly consistent return leg. As there may be more than one network that meets this consistency requirement, I want one that does so and maximises the minimum magnitude (absolute value) of the difference between ACM $x-$values of successive nodes in the ring. Work out the relevant magnitude and get back to me."

Now it's up to you! You can assume that links on one leg can cross links or nodes on the other leg without interference. (The third sample input illustrates a case of one leg crossing the other.) Although this non-interference would actually be achieved by having things at different heights, you can treat this as a 2D problem.

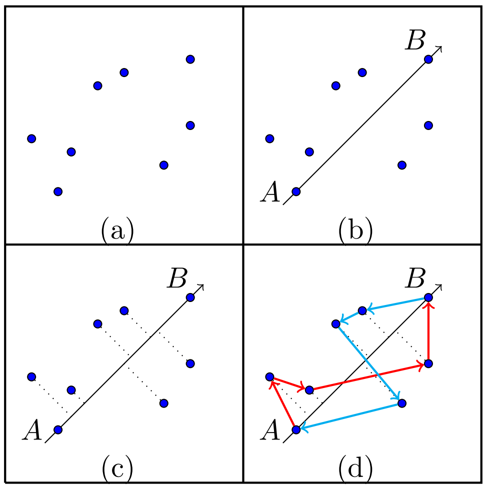


Figure 1: (a) The set of points; (b) Pick A and B (this also defines the ACM $x-$axis); (c) Each point has an $x-$value based on the ACM $x-$axis; (d) One example of a valid ring.

Input

The input will contain a single test case.

The input will begin with a line containing the number of potential locations, N ($3 \leq N \leq 100\,000$). The next N lines each contain two integers: the $x-$ and $y-$coordinates of each potential location ($-10^9 \leq x, y \leq 10^9$). These N points will be distinct. The potential locations are such that the opposition's network would involve at least 3 and at most 30 nodes.

Output

Output one real number being the value that the Communications Minister seeks. Any value that is within 10^{-3} of the correct answer will be considered correct.

Sample Input and Output

Sample Input 1	**Output for Sample Input**

4
0 0
4 0
0 4
4 4
	2.828427

Sample Input 2	**Output for Sample Input**

5
0 1
3 0
3 1
3 2
6 1
	0

Sample Input 3	**Output for Sample Input**

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

B: Banking II

Time Limit: 1 second(s)

A month ago at the South Pacific Divisional Contest, each of you (or one of your teammates) solved a problem about authenticating users. To refresh your memory:

The Actuarial Commerce Merchant bank has a scheme where, when you login, you are provided with a "pattern word", containing only upper and lower case letters. You must use this pattern word to extract and sum digits from your PIN as follows.

Letters in the pattern word are to be interpreted as numbers, with `a` (or `A`) $= 1$, `b` (or `B`) $= 2$, $\ldots$, `z` (or `Z`) $= 26$. A lower case letter specifies a count of digits to extract from the PIN while an upper case letter specifies a count of digits to be skipped. The letters in the pattern word are processed from left to right resulting in a sequence of extracted digits, which are added together to yield a number. You then enter that number into a field on the web page form to authenticate yourself. For example, if your PIN was `1093373`, and the pattern provided to you was `aBcA` you would extract one digit (namely `1`) skip two digits (`09`), extract 3 digits (`337`) and then skip 1 digit (`3`), before totalling the extracted digits (`1337`) and entering 14 into the field on the web page form.

The bank allows you to have a PIN containing up to 256 digits and they provide a pattern word in which the letters, when interpreted as numbers, sum to the length of the PIN.

Between the Divisional contest and now, something terrible has happened: someone has hacked into the bank's database and deleted all of the capital letters from every pattern word! In a panic, the bank has called you and requires your help again! For a given partial PIN, they would like to know the maximum value that could have been output from the algorithm mentioned above over all possible placements of capital letters such that the length of the pattern (when interpreted as numbers) matches the length of the PIN. The order of the lowercase letters may not be changed.

Input

The input will contain a single test case.

The first line of input will contain an n-digit PIN ($6 \leq n \leq 256$). The second line will contain an m-digit pattern word containing only lower case letters ($0 \leq m \leq n$).

Output

Output the maximum possible sum of extracted digits from the PIN. You may assume that at least one valid pattern exists (the bank had removed all of the bad cases after your help at Divisionals).

Sample Input and Output

Sample Input 1	Output for Sample Input

`092384907653`
`bc`
	`32`

Sample Input 2	Output for Sample Input

`092384907653`
`bb`
	`26`

This Page Is Intentionally (Almost) Blank.

C: Calculating Taxes

Time Limit: 8 second(s)

The Jones family is a very wealthy family who has lived in an area of the country for many generations. In this area of the country, houses are scattered around with dirt roads connecting them. Two houses are called *neighbours* if there is a dirt road connecting them. Using the dirt roads, there is exactly one way to get from any house to any other house without walking on some road at least twice. There is a new king in the country and he has decided that he will be taking taxes in a rather weird way.

Each household is to pick some divisor of their income from the last year (this value could be their full income if they wish). If any neighbours pick a number whose greatest common divisor is larger than 1, then the king will take every dollar from every person in the country. On the other hand, if each pair of neighbours pick numbers whose greatest common divisor is 1, then each household may keep whatever number they chose.

Obviously, one option is for each household to take $1 each, but that is not very profitable. Can you help them maximize the total amount of money that they can keep?

Input

The input will contain a single test case.

The first line will contain one integer n ($2 \leq n \leq 250$) denoting the number of houses. The next n lines contain the information about each household. The first line will contain information about house 1, the second line will contain information about house 2, etc.

Each of these lines will begin with two integers, I_k ($1 \leq I_k \leq 200\,000\,000$) and m_k ($1 \leq m_k \leq n-1$), denoting the income of the kth household and the number of neighbours of the kth household. Then follow m_k distinct integers denoting the neighbours of the kth household (each will be between 1 and n, inclusive, and none of these will be k).

Each road is bidirectional and will be listed twice in the input (once for each endpoint of the road).

Output

Output one integer, the maximum total amount of money that the households can keep.

Sample Input and Output

Sample Input 1	Output for Sample Input
4	10
2 1 2	
3 2 1 3	
2 2 2 4	
3 1 3	

Sample Input 2	Output for Sample Input

```
4
3 1 2
9 2 1 3
3 2 2 4
9 1 3
```
	`20`

Sample Input 3	Output for Sample Input

```
6
4 3 3 4 6
3 1 3
18 3 1 2 5
6 1 1
9 1 3
10 1 1
```
	`37`

D: Dropped Water Bottle

Time Limit: 2 second(s)

Tramper Joe is on a multi-day tramping trip. While he is wading across a river, his empty water bottle falls from his pack and starts bobbing off downstream. Joe watches it for some time, wondering if it's worth his while to chase after it. He eventually decides it is worthwhile so he returns to the left bank of the river, puts his pack down and starts to run downstream after the bottle. It has taken him 2 minutes to decide to pursue the bottle and get to the left bank. The bottle is now 60 metres downstream since the river is flowing at a steady 0.5 metres/second.

On the left bank of the river, Joe can run at a steady speed of 2 metres/second but there are intermittent extended obstacles such as cliff faces right beside the river. However, when he meets an obstacle, he must cross to the right bank where the terrain is rougher and he can run at only 1 metre/second. Crossing the river, which he always does on a line perpendicular to the river, takes 40 seconds. The right bank also has intermittent obstacles at which he must cross back again to the left bank.

If Joe gets to a point where there are obstacles on both banks of the river at the same point, Joe must give up and forfeit his water bottle. However, if an obstacle on one bank starts at the point where an obstacle on the opposite bank finishes, Joe is deemed able to cross from the first bank to the second, resuming his downstream run at exactly the same distance downstream as where the second obstacle finished. If both banks are clear, Joe must decide whether to stay on whatever side he is currently or to wade across the river and continue on the other side.

When Joe has overhauled the bottle by a sufficient amount, he wades into the centre of the river, which takes 20 seconds, and picks up his bottle.

Compute the minimum time for Joe to pick up his water bottle (if possible).

Input

The input will contain a single test case.

The first line contains two integers n_l ($0 \leq n_l \leq 2\ 000$) and n_r ($0 \leq n_r \leq 2\ 000$), the number of obstacles on the left bank and the right bank, respectively. The next n_l lines each contain two integers d_i ($0 < d_i \leq 6\ 000$) and ℓ_i ($0 < \ell_i \leq 6\ 000$), which are the distance downstream from where the bottle was dropped to the ith left-bank obstacle and the length of that obstacle respectively, both in metres. The following n_r lines similarly define the positions and lengths of all right-bank obstacles. On each bank, the obstacles will be given in increasing order based on their d_i value and do not overlap, though they may touch.

Output

Output the minimum time in seconds for Joe to reach his water bottle from when he starts running or the word `IMPOSSIBLE` if he cannot retrieve his bottle. Output will be considered correct if it is within 10^{-3} of the true answer.

Sample Input and Output

Sample Input 1	Output for Sample Input

3 1
2 1
5 1
165 1
3 2
	388

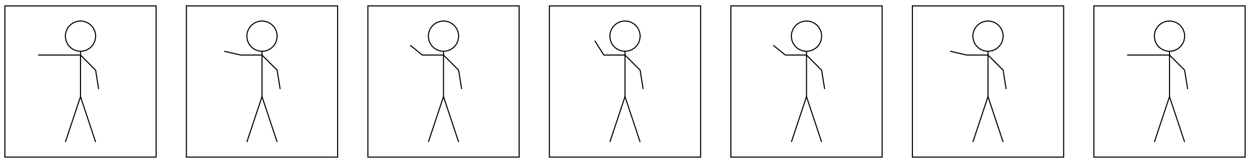
Sample Input 2	Output for Sample Input

1 1
3 1
3 1
	IMPOSSIBLE

E: Earl's Extremely Efficient Encryption

Time Limit: 2 second(s)

Earl has just finished writing an extremely efficient encryption algorithm for decreasing the size of video files. The algorithm has the capability to decrease the file size of a long video by many orders of magnitude. You can think of a video as a series of images.



One simple algorithm is to store each of these images individually. The secret of Earl's algorithm is to only use the differences between successive images in the series. He stores each transition in a 32-bit integer (these values are called the transition integers). Using an unencrypted version of the first image and all of the transition integers, you can fully recreate the video.

The issue is that calculating the transition integers is *lossy* (i.e. the 32-bit integer representation of the transition is only an approximation of the actual differences of the images). This makes it possible for errors to occur and the image to display incorrectly. Each transition on its own has a very high probability of working correctly, but the probability of being able to play the whole video without an error occurring is quite low. Moreover, the errors compound onto one another since the differences between the images in the actual video may not make sense for any images that Earl's algorithm has generated incorrectly.

To account for this issue, the algorithm will not only store the first image of the video, but a subset of the images. At each transition, the algorithm will check if the true (unencrypted) image is stored. If it is, then that image is displayed on the screen. If it is not stored, then the algorithm will use the transition integer to generate the next image.

The *badness* of a video is defined as the largest number of consecutive transition integers used. Earl's algorithm works for all videos consisting of at most L images. Given the number of images in the video collection and the subset of images Earl will include unencrypted, what is the badness value for each video?

Input

The input will contain a single test case.

The input will begin with six integers: k ($1 \le k \le 100$), n ($1 \le n \le 10^5$), L ($1 \le L \le 10^9$), a ($0 \le a \le L$), b ($0 \le b \le L$) and g_1 ($0 \le g_1 \le L$). We then define $g_0 = 0$ and

$$g_i = (a \cdot g_{i-1} + b) \pmod{L+1} \text{ for all } i \in \{2, \dots, n\}.$$

This is followed by k lines. The jth of these lines contains a single integer, w_j ($1 \le w_j \le L$), which represent the number of images in the jth video when it is stored uncompressed. The subset of images Earl includes unencrypted is

$$\{g_i : 0 \le i \le n\} \cap \{0, 1, \dots, w_j - 1\}.$$

The videos will be given in increasing order with respect to w_j.

Output

Output the badness for each video.

Sample Input and Output

Sample Input 1	Output for Sample Input

`4 5 20 1 2 2`
`1`
`3`
`7`
`14`
	`0`
	`1`
	`1`
	`3`

Sample Input 2	Output for Sample Input

`1 4 10 2 0 1`
`9`
	`3`

Explanation of Sample Input 1:

In the first sample input, the unencrypted images included are 0, 2, 4, 6, 8 and 10. For the video of length 1, you do not need any transition integers, so the badness is 0. For the video of length 3, you only need a transition integer to get from image 0 to image 1. For the video of length 7, you need to use a single transition integer on 4 separate occasions ($0 \to 1, 2 \to 3, 4 \to 5, 6 \to 7$). For a video of length 14, you need to use a transition integer to generate image 11, 12 and 13 (which corresponds to a badness of 3).

F: Flipping Switches

Time Limit: 7 second(s)

Having recently moved into a new home, you discover that its wiring is a bit messed up. A number of lights are controlled by a number of switches in a peculiar way. Every switch has two states: up and down (there is no use calling them *on* and *off*). Some experiments reveal that each light depends on three switches, at least one of which must be in the correct state for the light to shine. Of course, the opposite state of the same switch might be correct for another light.

Your first impulse was to set the switches so that as many lights as possible shine. However, your cook (who is an expert in circuitry) seems to have the opinion that this would be quite hard, so you settle for a more modest aim. Set the switches to a state so that flipping any single switch to its opposite state does not make more lights shine than before the flip.

Input

The input will contain multiple test cases.

The first line of input contains the number of test cases t $(1 \leq t \leq 25)$.

Each test case starts with a line containing the number of lights m $(1 \leq m \leq 4\ 000)$ and the number of switches n $(3 \leq n \leq 4\ 000)$. Each of the next m lines contains three distinct integers s_1, s_2 and s_3 $(1 \leq s_i \leq n)$, each of which is preceded by a `+` or `-` character. Line i gives the switches that control light i and their correct positions, where `+` means up and `-` means down. Light i will shine if at least one of the specified switches is set to its correct position.

Output

Output one line with a setting of the switches so that flipping any one of them does not make more lights shine. Give the setting as a string of n characters, each of which is `+` or `-`. Character i specifies the state of switch i. If more than one setting satisfies the constraints, display any such setting.

Sample Input and Output

Sample Input 1	Output for Sample Input
`2`	`+--`
`5 3`	`++++`
`+1 +2 +3`	
`+1 -2 +3`	
`-1 +2 +3`	
`-1 +2 -3`	
`-1 -2 +3`	
`4 4`	
`+1 +2 +4`	
`-1 -2 +4`	
`+2 +3 +4`	
`-2 -3 +4`	

This Page Is Intentionally (Almost) Blank.

G: Game Moves

Time Limit: 1 second(s)

2048 is a puzzle game where the objective is to slide numbered tiles on a grid to combine them into tiles with larger and larger values. While the initial aim of the game is to make a 2048 tile, the game continues until no more moves are possible, whether or not a 2048 tile has been made. The game starts with two tiles placed in random positions on the board, each having a value of either 2 or 4.

The game proceeds in a series of moves, with each move having five distinct phases (choice of direction; movement; merging; gap closing; birth) as follows:

1. Choice of direction: the player chooses one of the directions up, down, left or right. The chosen direction will be called "downstream".
2. Movement: the player moves all pieces downstream as far as they will go, closing any gaps. The edge of the board towards which all pieces move is called the "blocking edge".
3. Merging: working upstream, away from the blocking edge, any piece that has the same value as its upstream neighbour is merged with it, to form a new tile with the combined value of the two merging tiles. A merged tile is not a candidate for any further merging. Whenever a pair of tiles merge, the game score increases by the value of the new tile.
4. Gap closing: all pieces are again moved downstream as far as they will go to close any gaps.
5. Birth: a new tile with value 2 or 4 appears on an empty square on the board. The value (2 or 4) and the birth location are both random.

The board must change in some way during the second or third stage of this process for a new tile to appear for this to be considered a valid move.

The game score starts at zero and whenever two tiles combine the score increases by the value of the new tile. The game is usually played on a 4 × 4 grid but other variants are available where the grid is a different size but still square.

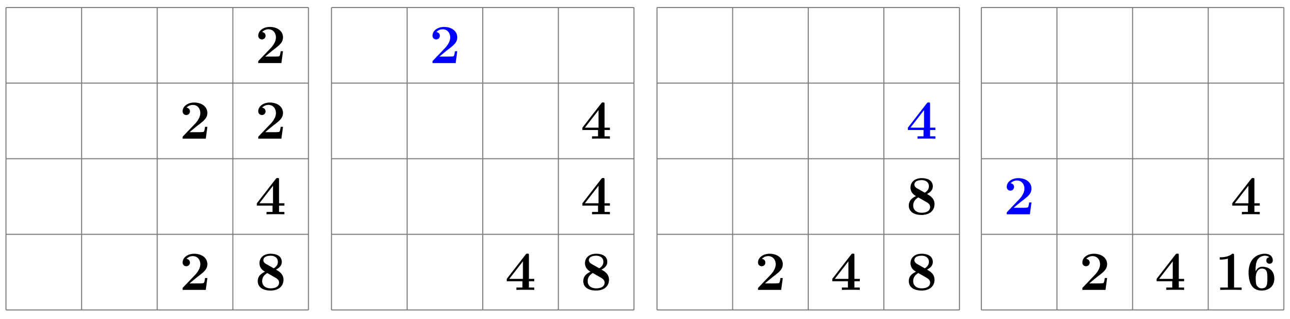
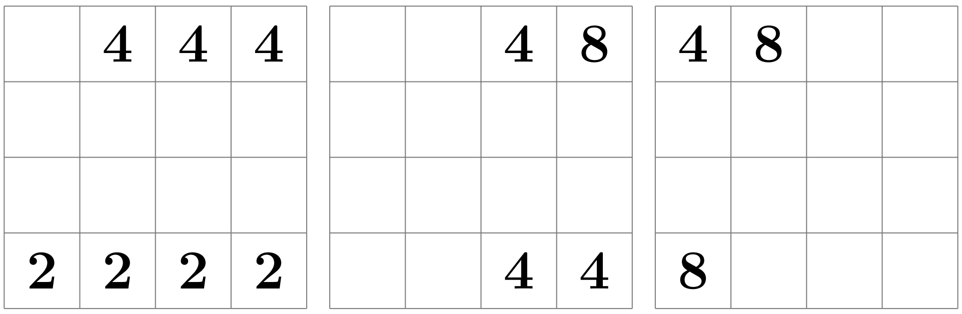

(a) Score is 16 (b) Down - Score is 24 (c) Down - Score is 32 (d) Down - Score is 48

Figure 1: Score progression for a series of moves - new values are shown in blue

Figure 1 shows a progression of moves with the score progressing from 16 to 24 to 32 to 48. The progression starts with Figure 1(a) which has been reached after seven moves. The player uses a Down move and the two vertical pairs of 2s in the third and fourth columns merge to become 4s. Because two 4s are generated, the score increases by 8. The resulting board is shown in Figure 1(b) with a new 2 randomly appearing in the second column. The player uses a second Down move and the vertical pair of

4s in the fourth column merge to become an 8. The score increases by the sum of the merged values (i.e. by 8). The board becomes that shown in Figure 1(c) with a new 4 randomly appearing in the fourth column. Finally, the player moves Down again and the vertical pair of 8s in the fourth column merge to become a 16, which increases the score by 16. The resulting board is shown in Figure 1(d).

Figure 2 is demonstrative and shows how values merge and move. This demonstration will not show any new values appearing on the board. Assuming the game board is in the state shown in Figure 2(a) then if the player moves Right, the two horizontal pairs of 2s in the bottom row merge to become 4s as shown in Figure 2(b), however only the rightmost pair of 4s in the top row merge to become an 8. If the player then moves left, the horizontal pair of 4s in the bottom row merge to become a single 8 as shown in Figure 2(c) and the values in the top row move but do not merge.



(a) Current State (b) After move Right (c) After move Left

Figure 2: How values merge and move

Given the state of a game and the current score, write a program to determine the number of moves it has taken to get to that state.

Input

The input will contain a single test case.

The first line of input for each test case is the size of the square grid, n ($2 \leq n \leq 7$). The following n lines contain n space separated integer values indicating the current state of a game. A value of 0 is used to indicate an empty grid square. All non-empty grid squares will have a value that is a power of 2 between 2 and 2^{50}, inclusive. This is followed by a line containing a single integer value being the score, s, for the current state of the game ($0 \leq s \leq 108\,086\,391\,056\,891\,712$).

All test cases will consist of a valid game state.

Output

Output the number of moves it has taken to achieve the current state of the game. The solution will have a value less than 2^{63}.

Sample Input and Output

Sample Input 1	**Output for Sample Input**

`4`
`0 0 2 4`
`2 0 8 8`
`0 4 8 16`
`4 8 16 256`
`1900`
	`150`

acm International Collegiate Programming Contest 2015 IBM event sponsor
SOUTH PACIFIC REGION

H: Hiking

Time Limit: 10 second(s)

Alice likes to hike. She likes some bits of track and dislikes others. A cost function will model her distaste for (or liking of) all available track sections. Given a map with the costs of going from one location to the next, find the average cost for a hike in that area.

The map is a rectangular grid of cells. Each cell has information about the cost of going from it to each of its northern, western, southern and eastern neighbours. A hike is a finite sequence of steps each of which goes from a cell to one of its neighbours. The cost of a hike is the sum of the costs of its steps.

When Alice hikes from a given start point to a given end point, she always chooses a minimal cost hike. Determine the average cost of such minimal-cost hikes taken over all possible start and end cells. The start and end cells of a hike must be different. Each pair of start and end cells counts just once towards the average even if there are several minimal-cost hikes between the two cells.

Input

The input will contain a single test case.

The first line contains two integers w and h specifying the width and the height of the map ($2 \le w, h \le 50$). Each of the following h lines contains w integers n_{ij} ($1 \le i \le h$ and $1 \le j \le w$ and $-10^7 \le n_{ij} \le 10^7$) specifying the cost of going from cell (i, j) one step to its northern neighbour $(i - 1, j)$. Then follow three blocks of h lines each, similarly specifying costs for the western, southern and eastern neighbours in this order.

It is not permitted to walk off the map; the corresponding costs will be 0. A minimal-cost hike will exist between any two cells.

Output

Display the smallest integer that is greater than or equal to the average cost of minimal-cost hikes among all pairs of distinct start and end cells.

Sample Input and Output

Sample Input 1	Output for Sample Input

```
2 2
0 0
-200 0
0 100
0 500
400 300
0 0
200 0
-400 0
```
	50

This Page Is Intentionally (Almost) Blank.

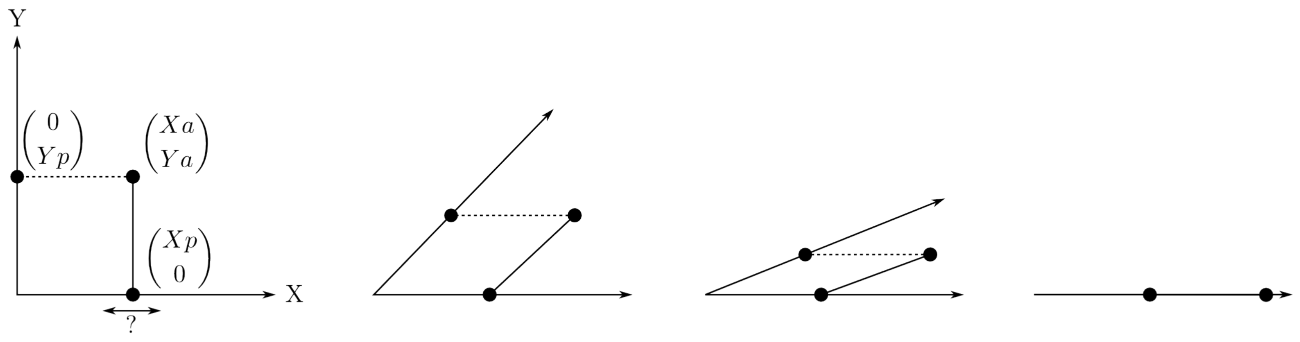
acm International Collegiate Programming Contest 2015 IBM event sponsor
SOUTH PACIFIC REGION

I: Individually Customised Pop-up Cards

Time Limit: 3 second(s)

The greeting card crisis is at hand, but have you noticed laser cutters becoming available everywhere? This means it's time for the pop-up card renaissance!

Making a card that folds can be time-consuming without software, but as you are about to prove, this is well within the reach of a blitz hackathon. Keeping it simple for the initial version, we are going to deal with four planar polygons, joined along four lines (hinges), which are parallel to each other at all times. Call the common direction of those hinges the z direction. Note that we can simply deal with the projection of this structure onto the $z = 0$ plane, and so we will omit the z coordinate from further discussion.


Y
$\begin{pmatrix} 0 \\ Yp \end{pmatrix}$
$\begin{pmatrix} Xa \\ Ya \end{pmatrix}$
$\begin{pmatrix} Xp \\ 0 \end{pmatrix}$
X
?

We start with the two pages of the card at 90° to each other, represented by the positive x and y axes. For simplicity, consider them to extend infinitely far. Our folding structure is then described by a point in the first quadrant and two line segments joining that point to each of the two pages. As the card folds, the two pages rotate towards each other and, if the line segments are chosen appropriately, the whole structure folds flat without getting stuck at any time. At the end of the folding process they should all become collinear. We do not allow either the pages or the line segments to bend, stretch, crease or become disconnected from their hinges. Also, the angle between the pages may never exceed 90°.

We let the user draw a single line segment connected to the x-axis, with the other end positioned anywhere in the first quadrant, as appropriate to their design concept. We then supply suggestions for the second line segment to support the first by connecting it with the other page while satisfying the folding requirement. If no options exist, we adjust the input by the smallest amount possible until at least one option becomes available. The particular quantity we are interested in now is the smallest distance to move the point where the user-supplied line segment contacts the x-axis.

Input

The input will contain multiple test cases.

Each test case appears on a line of its own and will consist of three integers, separated by single spaces: X_a ($1 \leq X_a \leq 2\ 000$), Y_a ($1 \leq Y_a \leq 2\ 000$) and X_p ($1 \leq X_p \leq 2\ 000$). These describe the line segment drawn by the user: from the point $(X_p, 0)$ to the point (X_a, Y_a).

The input is terminated by three space-separated zeros on a line. There are at most 1 500 test cases.

Output

For each test case, print a non-negative floating-point number on a line by itself, giving the smallest distance that X_p should be moved by, such that we could then add some line segment from (X_a, Y_a) to some $(0, Y_p)$ so the card will close flat. (Note that $Y_p > 0$, of course.)

Responses will be judged correct as long as they are within 10^{-3} of the true answer. As a technical note, to make this terminology precise, the true answer is defined as the smallest number such that valid distances may be found arbitrarily close to it (see 'Explanation of Sample Input' for an example of this).

Sample Input and Output

Sample Input 1	Output for Sample Input

`1 1 1`
`1 1 10`
`5 4 20`
`0 0 0`
	`0`
	`0`
	`7.5`

Explanation of Sample Input:

The first test case is shown in the diagram above. In this case there is no need to move X_p, because if we draw the segment from $(1, 1)$ to $(0, 1)$, dashed in the diagram, the result is a structure folding flat as shown.

In the third case, any distance *strictly* greater than 7.5 (and less than 20) leads to a valid card.

J: Jumping Impala

Time Limit: 8 second(s)

Vlad the impala is both excited and nervous, as in a few months he will be participating in the annual challenge event where his herd will choose its leader for the coming year. As the eldest, he will be the first to undertake the challenge and will become the new leader if he succeeds. If Vlad does not succeed, the second eldest will then undertake the challenge and so on.

The impalas want a leader who can lead them safely to grazing grounds. Along the way, there are many predators and the challenge event is designed to test skills needed to avoid predators. Past volcanic activity has created a crocodile infested circular lake with a lushly vegetated circular island located exactly on the lake's centre. The challenge is to go from the outer edge of the lake to the island, graze for a couple of hours, then come back to the outer edge and finally make a second trip out to the island and again return back to the outer edge. Impalas are so frightened by the crocodiles that if an impala touches the water, it freezes and awaits its fate. Fortunately, there are some (unit radius) circular stones in the lake that impalas may use. By using these, it is not necessary to go to the island in a single bound from the lake's outer edge. Unfortunately, not only can crocodiles grab an impala in the water, but they can also grab an impala from a stone (but not the island or outer edge of the lake). However, Vlad has noticed that the crocodiles, which are slow by comparison with a leaping impala, always position themselves to exploit the impala tendency to reuse a previous route, e.g. returning from grazing on the island via the same stones as used on the way to the island. Indeed, the crocodiles never grab an impala from a stone that it is visiting for the first time. Vlad aims to exploit this weakness in the crocodile strategy. (As the first to attempt the challenge, he's not in a position to become leader by the usual manner—being the lucky impala that happens to attempt the challenge when the crocodiles are too full of previous challenge participants to be interested in another!)

An impala can leap (up to) a certain distance any number of times and turn arbitrarily tightly, so Vlad is working on improving his maximum leap distance and wants to know what it needs to be for him to be able to go to and from the island twice in safety (i.e. without ever revisiting any stone or touching the water). As Vlad has failed to "get real", he wants to know the relevant integer value, e.g. if he needs to be able to leap at least 2.01, you'll have to tell him 3 to be on the safe side. You can assume that Vlad can leap from the exact edge of one object to the exact edge of another and that it is safe to leap over a previously visited stone without landing on it.

Input

The input will contain a single test case.

The input starts with a line containing the radius of the lake, L ($4 \leq L \leq 10^9$), the radius of the island, R ($1 \leq R \leq L - 3$), and the number of stones in the lake, S ($4 \leq S \leq 1\,000$). The next S lines contain the information about the stones, one stone per line. Each line describing a stone contains two integers, the $x-$ and $y-$coordinates of its centre, regarding the lake centre as the origin (each coordinate will have absolute value no more than 10^9). These stones will be fully in the water. Stones may touch each other, the island, or the lake's outer edge, but there will be no overlaps between these items.

Output

Output the integer value Vlad wants to know. You can be assured that Vlad has to jump a positive distance in order to complete the challenge—otherwise, it wouldn't be a challenge!

Sample Input and Output

Sample Input 1	Output for Sample Input

10 2 5
-6 0
0 -6
0 6
3 0
9 0
	4

Sample Input 2	Output for Sample Input

10 2 5
-6 0
0 -6
0 6
3 1
9 0
	5

K: Krypton Stadiums

Time Limit: 10 second(s)

The planet of Krypton contains n cities. These cities are located in distinct locations all along one straight line running west-east. The cities are labelled $0, 1, 2, \ldots, n-1$ in order from west to east. Each city is home to one team and one stadium. Each stadium will have two corresponding integers (a_i and b_i) defining an interval of cities that may play at that stadium. That is, a team from city x may only play at stadium i if $a_i \le x \le b_i$. It is guaranteed that every team will be able to play at its home stadium ($a_i \le i \le b_i$).

You have been hired to make the schedule for the upcoming season and must determine if the layout of the stadiums and cities is *great*, *acceptable* or *bad*.

- The layout is *great* if for every pair of cities, c_1 and c_2, there is a stadium in between c_1 and c_2 (inclusive) that can host the teams from both c_1 and c_2.
- The layout is *acceptable* if it is not *great*, but for every pair of cities, c_1 and c_2, there is some stadium that can host the teams from both c_1 and c_2.
- The layout is *bad* if there is some pair of cities where no stadium can host the teams from both cities.

Input

The input will contain multiple test cases.

The first line of each test case will contain an integer n ($2 \le n \le 200\,000$) denoting the number of cities. The next n lines will give the intervals of each stadium. The intervals are given by exactly 6 characters. The first three characters will denote a_i and the last three characters will denote b_i (whose definitions are given above). Each set of three characters will denote a base 62 number (using the ordering `0-9A-Za-z` as our alphabet). For example, cities 0, 1, 9, 10, 35, 36, 61, 62 and 199 999 are represented by `000`, `001`, `009`, `00A`, `00Z`, `00a`, `00z`, `010` and `q1n`, respectively.

Input will be terminated by end of file. There will be no more than 1 000 different test cases and there will be no more than 2 000 000 stadiums across all test cases.

Output

For each test case, output one of three strings: `Great`, `Acceptable` or `Bad`.

Sample Input and Output

Sample Input 1	Output for Sample Input
4 000001 000003 002002 002003 4 000000 001001 002002 003003 4 000001 000003 002002 003003	Great Bad Acceptable