Problem A+

Consanguine Calculations

Input File: blood.in

Every person's blood has 2 markers called ABO *alleles*. Each of the markers is represented by one of three letters: `A`, `B`, or `O`. This gives six possible combinations of these alleles that a person can have, each of them resulting in a particular ABO *blood type* for that person.

Combination	*ABO Blood Type*
AA	A
AB	AB
AO	A
BB	B
BO	B
OO	O

Likewise, every person has two alleles for the blood *Rh factor*, represented by the characters `+` and `-`. Someone who is "Rh positive" or "Rh+" has at least one `+` allele, but could have two. Someone who is "Rh negative" always has two `-` alleles.

The blood type of a person is a combination of ABO blood type and Rh factor. The blood type is written by suffixing the ABO blood type with the `+` or `-` representing the Rh factor. Examples include `A+`, `AB-`, and `O-`.

Blood types are inherited: each biological parent donates one ABO allele (randomly chosen from their two) and one Rh factor allele to their child. Therefore 2 ABO alleles and 2 Rh factor alleles of the parents determine the child's blood type. For example, if both parents of a child have blood type `A-`, then the child could have either type `A-` or type `O-` blood. A child of parents with blood types `A+` and `B+` could have any blood type.

In this problem, you will be given the blood type of either both parents or one parent and a child; you will then determine the (possibly empty) set of blood types that might characterize the child or the other parent.

Note: an uppercase letter "Oh" is used in this problem to denote blood types, *not* a digit (zero).

Input

The input consists of multiple test cases. Each test case is on a single line in the format: the blood type of one parent, the blood type of the other parent, and finally the blood type of the child, except that the blood type of one parent or the child will be replaced by a question mark. To improve readability, whitespace may be included anywhere on the line except inside a single blood type specification.

The last test case is followed by a line containing the letters `E`, `N`, and `D` separated by whitespace.

Output

For each test case in the input, print the case number (beginning with 1) and the blood type of the parents and the child. If no blood type for a parent is possible, print "IMPOSSIBLE". If multiple blood types for parents or child are possible, print all possible values in a comma-separated list enclosed in curly braces. The order of the blood types inside the curly braces does not matter.

The sample output illustrates multiple output formats. Your output format should be similar.

Sample Input	Output for the Sample Input
O+ O- ? O+ ? O- AB- AB+ ? AB+ ? O+ E N D	Case 1: O+ O- {O+, O-} Case 2: O+ {A-, A+, B-, B+, O-, O+} O- Case 3: AB- AB+ {A+, A-, B+, B-, AB+, AB-} Case 4: AB+ IMPOSSIBLE O+

Problem B

Containers

Input File: containers.in

A seaport container terminal stores large containers that are eventually loaded on seagoing ships for transport abroad. Containers coming to the terminal by road and rail are stacked at the terminal as they arrive.

Seagoing ships carry large numbers of containers. The time to load a ship depends in part on the locations of its containers. The loading time increases when the containers are not on the top of the stacks, but can be fetched only after removing other containers that are on top of them.

The container terminal needs a plan for stacking containers in order to decrease loading time. The plan must allow each ship to be loaded by accessing only topmost containers on the stacks, and minimizing the total number of stacks needed.

For this problem, we know the order in which ships must be loaded and the order in which containers arrive. Each ship is represented by a capital letter between A and Z (inclusive), and the ships will be loaded in alphabetical order. Each container is labeled with a capital letter representing the ship onto which it needs to be loaded. There is no limit on the number of containers that can be placed in a single stack.

Input

The input file contains multiple test cases. Each test case consists of a single line containing from 1 to 1000 capital letters representing the order of arrival of a set of containers. For example, the line ABAC means consecutive containers arrive to be loaded onto ships A, B, A, and C, respectively. When all containers have arrived, the ships are loaded in strictly increasing order: first ship A, then ship B, and so on.

A line containing the word `end` follows the last test case.

Output

For each input case, print the case number (beginning with 1) and the minimum number of stacks needed to store the containers before loading starts. Your output format should be similar to the one shown here.

Sample Input	Output for the Sample Input
A CBACBACBACBACBA CCCCBBBBAAAA ACMICPC end	Case 1: 1 Case 2: 3 Case 3: 1 Case 4: 4

This page intentionally left blank.

Problem C

Grand Prix

Input File: gp.in

You are the chief designer of a road race that will be held in a hilly area. The racecourse consists of N connected straight line segments; that is, the end of the kth segment coincides with the beginning of the $(k+1)$st segment. During planning, these segments are laid out on a planar surface, so 2-dimensional Cartesian coordinates identify the endpoints. The first segment begins at the origin. Figure 1 shows the planar view of a racecourse with 6 segments.

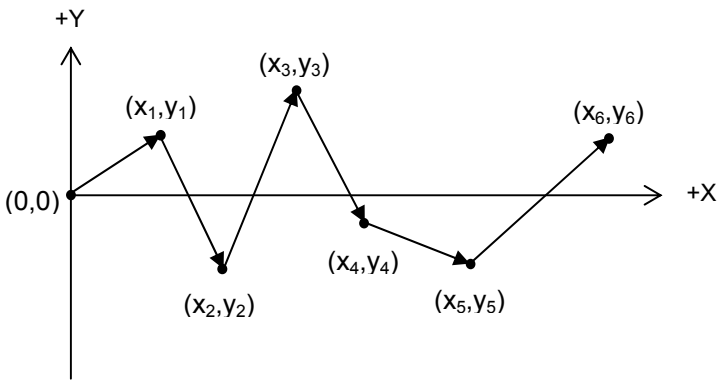


Figure 1

The actual race is run on the side of a hill. For simplicity, assume the hill makes an angle θ with the horizontal plane, as illustrated in Figure 2.

The plane of the hill and the horizontal plane intersect in the y-axis. Each 2-dimensional Cartesian coordinate (x_i, y_i) corresponds to a 3-dimensional form (x_i', y_i', z_i'), with z_i' representing the height of the endpoint of the ith linear segment. The height of the origin is 0.

This particular race is intended for novice drivers, so the racecourse must not include any segments that require downhill travel. That is, if the height of the endpoint of segment k is z_k', then the height of the endpoint of each segment after segment k must not be less than z_k'. Formally we can write $z_k' \le z_m'$, for $m \ge k$.

If a proposed racecourse includes downhill segments, it might be possible to transform it into a racecourse with no downhill segments by rotating the planar view of the entire course about the origin, without changing the angle between consecutive pairs of segments. However there may be proposed racecourses that cannot be made acceptable by such a rotation.

In this problem you must determine if a proposed racecourse is acceptable (that is, if it does not contain any downhill segments). If it is not acceptable, you must determine the minimum angle through which the racecourse must be rotated to make it acceptable, if that is possible.

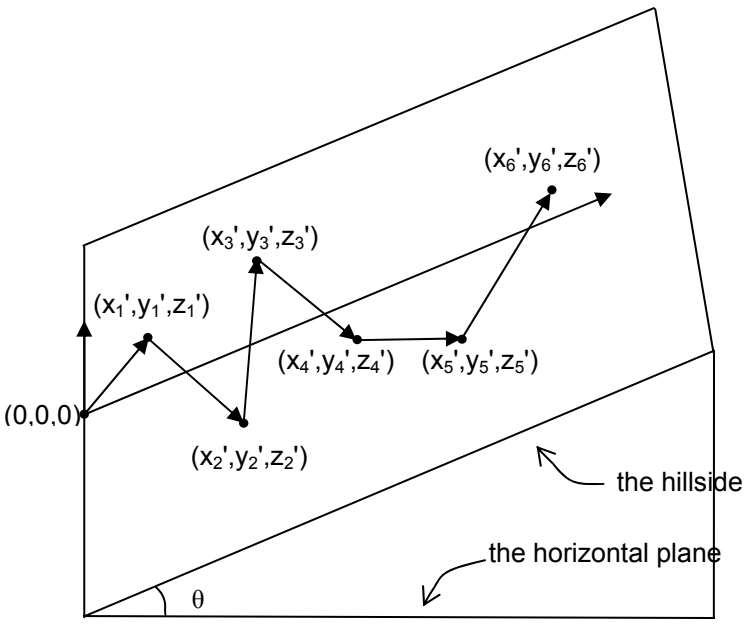


Figure 2

Input

The input consists of multiple test cases. Each test case is a description of a proposed racecourse and the slope of the hillside on which it will be run. The first line of each description contains two integers N ($1 \le N \le 10000$) and θ ($0^\circ \le \theta \le 45^\circ$). N denotes the number of segments in the course and θ denotes the angle (in degrees) that the hillside makes with the horizontal plane. Each of the next N lines contains a pair of integers (x_i, y_i), ($1 \le i \le N$), which are the endpoints of the linear segments comprising the racecourse. The first segment begins at the origin, and segment $k+1$ begins at the endpoint of segment k. No segment has zero length.

The last test case is followed by a line containing two zeroes.

Output

For each test case, print a line containing the test case number (beginning with 1). If the proposed course is acceptable without rotation, print "Acceptable as proposed". If the course is not acceptable as proposed, but can be made acceptable by rotating it about the origin, print "Acceptable after clockwise rotation of X degrees" or "Acceptable after counterclockwise rotation of X degrees". The value X should be an unsigned number. For our purposes, a clockwise rotation would rotate the positive y-axis toward the positive x-axis. If both a clockwise and a counterclockwise rotation can make the course acceptable, choose the one with the smaller angle. If both rotations have the same angle, then either a clockwise or counterclockwise rotation will be accepted. If the course cannot be made acceptable by any rotation, print "Unacceptable". Display the angles of rotation rounded to two fractional digits.

Print a blank line after the output for each test case. Use an output format similar to that shown in the sample output below.

Sample Input	Output for the Sample Input
2 45 10 10 0 1 2 0 1 1 2 0 3 45 10 10 1 10 5 6 0 0	Case 1: Acceptable after clockwise rotation of 131.99 degrees Case 2: Acceptable as proposed Case 3: Unacceptable

Problem D
Jacquard Circuits
Input: jacquard.in

The eccentric sculptor Albrecht Caravaggio Mondrian has been inspired by the history of the computer to create works of art that he calls "Jacquard circuits." Each of his works of art consists of a series of polygonal circuit boards (defined below), all having the same shape but at different scales, joined together with wire or string into a three-dimensional tiered structure. Figure 1 below shows an example with two levels and a pentagonal shape.

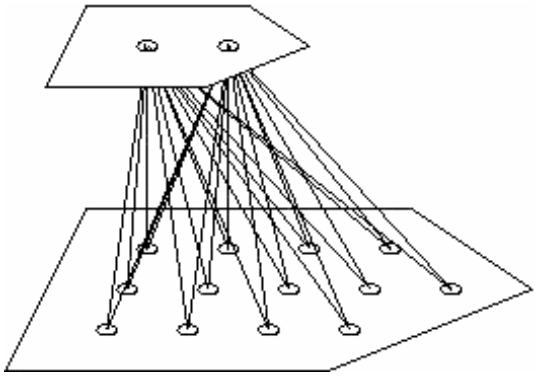


Figure 1

A.C.M. (as he is known to his friends) bases his art upon the punched hole cards of the Jacquard loom (that were later adapted by Charles Babbage for his analytical engine) and upon the regular grid layout approach often used in circuit interfacing (for instance, Pin Grid Arrays).

The circuit boards used in the sculptures are in the shapes of lattice polygons. A lattice polygon is defined as any closed, non-self-intersecting cycle of straight line segments that join points with integer coordinates, with no two consecutive line segments parallel. For any given lattice polygon *P*, there is a smallest lattice polygon with the same shape and orientation as *P* – call this the *origin*. Smaller polygons of the same shape and orientation as *P* are called its predecessors and polygons larger than *P* are its successors. (See Figure 2 on the following page.)

To build one of his sculptures, A.C.M. begins by randomly selecting *N* lattice points and then drawing a pattern by connecting these points. (Note that not all of the *N* points are necessarily vertices of a lattice polygon. This may happen, for example, if three or more consecutive points are collinear.) Let *P* be the lattice polygon determined by the pattern. Then he determines the origin corresponding to *P*, selects the number of levels, *M*, he wishes to use, and constructs the first *M* polygonal circuit boards in the series (that is, the origin and its first *M*-1 successors). Each lattice point lying strictly within the boundary of any of these polygons is a hole where strings or wires meet.

The hard part of creating the sculpture is tying together all the strings or wires that meet at a given hole. Furthermore, some of A.C.M.'s famous miniaturized sculptures, built using nano-engineering techniques, involve hundreds of thousands of levels. Mondrian would like a way to determine, given a polygonal shape and the number of levels, how many holes there will be in the final sculpture. You must write a program to help him. (For example, the sculpture in Figure 1 above has 15 holes.) Assume holes have zero diameter.

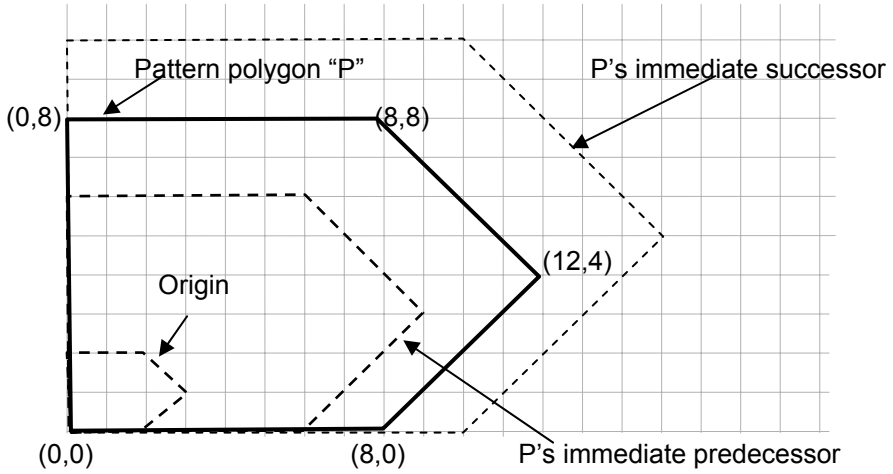


Figure 2

Input

The input consists of one or more test cases. Each case begins with a line containing two positive integers N ($3 \le N \le 1000$) and M ($1 \le M \le 1000000$). N is the number of lattice points Mondrian selected and M is the number of levels in the completed sculpture. Each of the next N lines contains two integers x, y ($|x|$, $|y| \le$ 1000000) which denote the coordinates of one of Mondrian’s points. Points are listed in either clockwise or counterclockwise order.

The last test case is followed by a line containing two zeroes.

Output

For each test case, print a line containing the test case number (beginning with 1) followed by the number of holes in an M-level sculpture starting from the origin polygon of the given pattern. In no case will this value exceed the maximum possible value of a 64-bit signed integer.

Sample Input	Output for the Sample Input
5 2 0 0 8 0 12 4 8 8 0 8 3 2 -1 -1 3 1 5 -1 0 0	Case 1: 15 Case 2: 2

Problem E

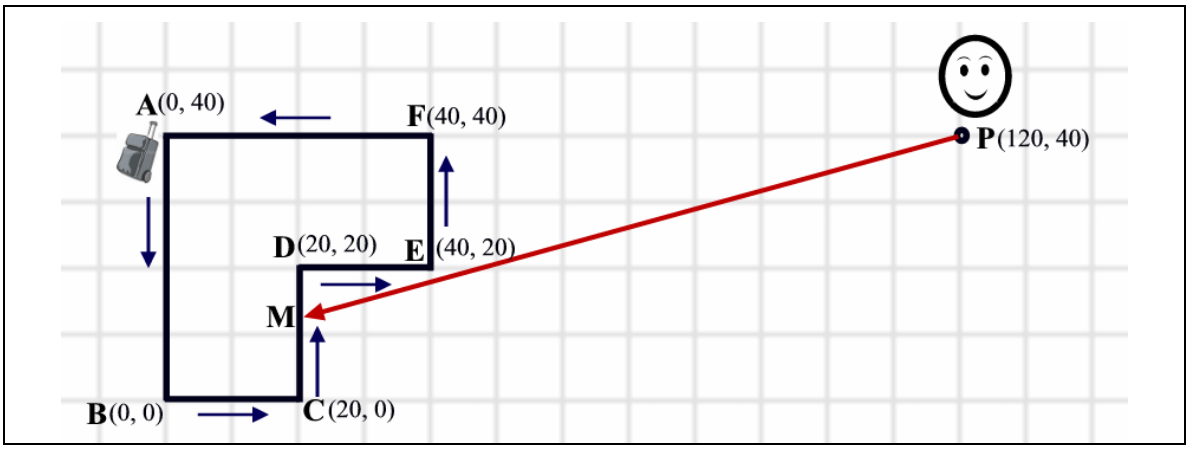
Collecting Luggage

Input: luggage.in

Collecting your luggage after a flight can be far from trivial. Suitcases and bags appear on a conveyor belt, and hundreds of passengers fight for a good vantage point from which to find and retrieve their belongings. Recently, the Narita Airport Authority has decided to make this process more efficient. Before redesigning their baggage claim areas, they need a simulation program to determine how average passengers behave when collecting their luggage. This simulation assumes that passengers will always take a path of straight line segments to reach their luggage in the least amount of time.

For this problem, a conveyor belt is modeled as a simple polygon. A luggage piece appears on some point of the conveyor belt, and then moves along the conveyor belt at a constant speed. A passenger is initially positioned at some point outside the conveyor belt polygon. As soon as the piece of luggage appears, the passenger moves at a constant speed (which is greater than the speed of the luggage piece) in order to pick up the luggage. The passenger's path, which may not cross over the conveyer belt but may touch it, puts the passenger in the same position as the moving piece of luggage in the least amount of time.

In the following figure, the conveyor belt is depicted as a polygon ABCDEF. The luggage starts at the top-left corner (Point A) and moves in counterclockwise direction around the polygon as shown with the small arrows. The passenger begins at point P and moves on the path that puts him and the luggage into the same place (point M in the figure) in the shortest amount of time. The passenger's path is shown by a red arrow. This figure corresponds to the first sample input.



Input

The input consists of one or more test cases describing luggage pickup scenarios. A scenario description begins with a line containing a single integer N ($3 \le N \le 100$), the number of vertices of the conveyor belt polygon. This is followed by N lines, each containing a pair of integers x_i, y_i ($|x_i|, |y_i| \le 10000$) giving the coordinates of the vertices of the polygon in counterclockwise order. The polygon is simple, that is, it will not intersect itself and it will not touch itself. The polygon description is followed by a line containing two integers p_x, p_y ($|p_x|, |p_y| \le 10000$), the coordinates of the starting position of the passenger. The last line of the description contains two positive integers V_L and V_P ($0 < V_L < V_P \le 10000$), which are the speed of the luggage and the passenger respectively. All the coordinates are given in meters, and the speeds are given in meters per minute.

You can assume that the passenger is positioned outside the conveyor belt polygon. The luggage will move in counterclockwise direction around the conveyor belt, starting at the first vertex of the polygon.

There is a blank line after each test case in the input file.

Output

For each test case, print a line containing the test case number (beginning with 1) followed by the minimum time that it takes the passenger to reach the luggage. Use the formatting shown in the sample output (with minutes and seconds separated by a colon), rounded to the nearest second. The value for seconds should have two digits after the decimal point and be printed in a field width of two (padded with leading zeroes if required).

Sample Input	Output for the Sample Input
6 0 40 0 0 20 0 20 20 40 20 40 40 120 40 70 100 4 0 0 10 0 10 10 0 10 100 100 10 11 0	Case 1: Time = 1:02 Case 2: Time = 12:36

Problem F

Marble Game

Input File: marble.in

A Marble Game is played with *M* marbles on a square board. The board is divided into *N*x*N* unit squares, and *M* of those unit squares contain holes. Marbles and holes are numbered from 1 to *M*. The goal of the Marble game is to roll each marble into the hole that has the same number.

A game board may contain walls. Each wall is one unit long and stands between two adjacent unit squares. Two squares are considered adjacent if and only if they share a side.

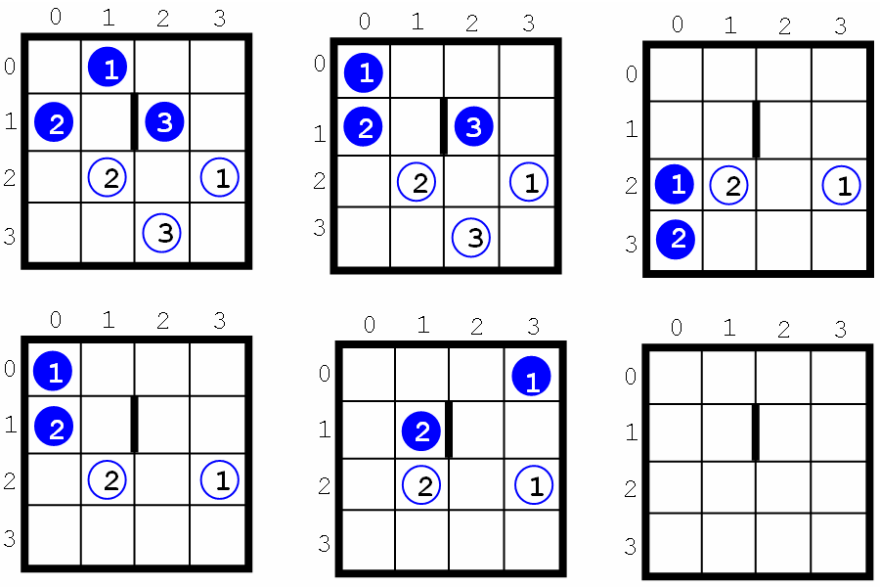
At the beginning of the game, all marbles are placed on the board, each in a different square. A "move" consists of slightly lifting a side of the game board. Then all marbles on the board roll downward toward the opposite side, each one rolling until it meets a wall or drops into an empty hole, or until the next square is already occupied by another marble. Marbles roll subject to the following restrictions:

- Marbles cannot jump over walls, other marbles, or empty holes.
- Marbles cannot leave the board. (The edge of the board is a wall.)
- A unit square can contain at most a single marble at any one time.
- When a marble moves into a square with a hole, the marble drops into that hole. The hole is then filled, and other marbles can subsequently roll over the hole. A marble in a hole can never leave that hole.

The game is over when each marble has dropped into a hole with the corresponding number.

The figure below illustrates a solution for a game played on a 4x4 board with three blue marbles, three holes and a wall. The solution has five moves: lift the east side, lift the north side, lift the south side, lift the west side, lift the north side.

Your program should determine the fewest number of moves to drop all the marbles into the correct holes – if such a move sequence is possible.



Input

The input file contains several test cases. The first line of each test case contains three numbers: the size N ($2 \le N \le 4$) of the board, the number M ($M > 0$) of marbles, and the number W of walls. Each of the following $2M$ lines contains two integers. The first integer is a row location and the second is a column location. The first M of those lines represent the locations of the marbles, where marble#1 is on the first line, marble#2 on the second, and so on. The last M of those lines represent the locations of the holes, with the location of hole#1 coming first, hole#2 coming second, and so on. Finally, the next W lines represent the wall locations. Each of those lines contains four integers: the first pair are the row and column of the square on one side of the wall and the second pair are the row and column of the square on the other side of the wall. Rows and columns are numbered 0..N-1.

The input file ends with a line containing three zeroes.

Output

For each test case, print the case number (beginning with 1) and the minimal number of moves to win the game. If the game cannot be won, print the word "impossible". Put a blank line after each test case. Use the format of the sample output below.

Sample Input	Output for the Sample Input
4 3 1 0 1 1 0 1 2 2 3 2 1 3 2 1 1 1 2 3 2 2 0 0 0 1 0 2 2 0 2 0 1 0 2 0 2 1 0 0 0	Case 1: 5 moves Case 2: impossible

Problem G

Network

Input file: network.in

A packet-switching network handles information in small units, breaking long messages into multiple packets before routing. Although each packet may travel along a different path, and the packets comprising a message may arrive at different times or out of order, the receiving computer reassembles the original message correctly.

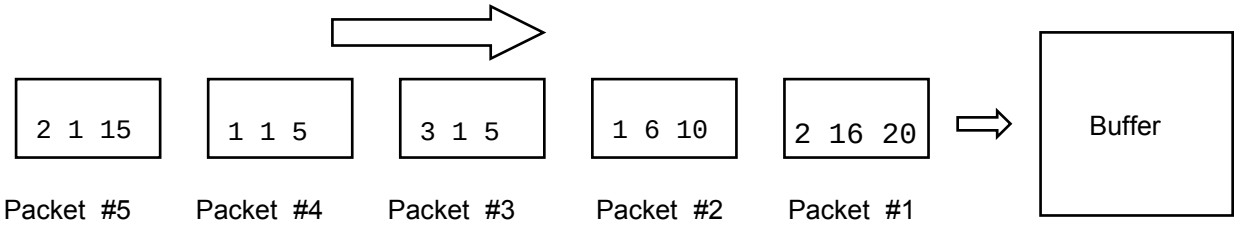
The receiving computer uses a buffer memory to hold packets that arrive out of order. You must write a program that calculates the minimum buffer size in bytes needed to reassemble the incoming messages when the number of messages (N), the number of packets (M), the part of the messages in each packet, the size of each message, and the order of the incoming packets are given.

When each packet arrives, it may be placed into the buffer or moved directly to the output area. All packets that are held in the buffer are available to be moved to the output area at any time. A packet is said to "pass the buffer" when it is moved to the output area. A message is said to "pass the buffer" when all of its packets have passed the buffer.

The packets of any message must be ordered so the data in the sequence of packets that pass the buffer is in order. For example, the packet containing bytes 3 through 5 of a message must pass the buffer before the packet containing bytes 6 through 10 of the same message. Messages can pass the buffer in any order, but all packets from a single message must pass the buffer consecutively and in order (but not necessarily at the same time). Note that unlike actual buffering systems, the process for this problem can look ahead at all incoming packets to make its decisions.

The packets consist of data and header. The header contains three numbers: the message number, the starting byte number of data in the packet, and the ending byte number of data in the packet respectively. The first byte number in any message is 1.

For example, the figure below shows three messages (with sizes of 10, 20, and 5 bytes) and five packets. The minimum buffer size for this example is 10 bytes. As they arrive, packet #1 and packet #2 are stored in the buffer. They occupy 10 bytes. Then packet #3 passes the buffer directly. Packet #4 passes the buffer directly and then packet #2 exits the buffer. Finally, packet #5 passes the buffer directly and packet #1 exits the buffer.



Input

The input file contains several test cases. The first line of each test case contains two integers N ($1 \le N \le 5$) and M ($1 \le M \le 1000$). The second line contains N integers that are the sizes of messages in bytes; the first number denotes the size of message #1, the second number denotes the size of message #2, and so on. Each of the following M lines describes a packet with three integers: the message number and the starting and ending byte numbers of data in the packet. No packet contains more 64 bytes of data.

The last test case is followed by a line containing two zeroes.

Output

For each test case, print a line containing the test case number (beginning with 1) followed by the minimum buffer size in bytes required to reassemble the original messages. Put a blank line after the output for each test case. Use the format of the sample output.

Sample Input	Output for the Sample Input
3 3 5 5 5 1 1 5 2 1 5 3 1 5 3 5 10 20 5 2 16 20 1 6 10 3 1 5 1 1 5 2 1 15 0 0	Case 1: 0 Case 2: 10

Problem H

Raising the Roof

Input File: roof.in

You have agreed to help a roofer estimate the amount of material required to protect the surface of a roof from sun damage by computing the surface area of the roof that can be seen when viewed from directly above.

The geometry of a roof is defined by the vertices of the triangles that enclose the plane surfaces of the roof. The x and y coordinates of a vertex are determined from an aerial view of the roof, and the z coordinate is the height of the vertex above the ground, assuming that the ground is level (which we do). For example, a line segment with vertices (10,10,10) and (10,20,10) is parallel to the ground, and is 10 units long. A line segment with vertices (10,10,10) and (10,18,16) slopes from 10 units to 16 units above the ground; in the aerial view of the roof it is 8 units long, but the actual length is 10 units.

In an aerial view of a roof, one region of the roof can sometimes obscure a lower region. Only visible portions of the roof should be included in the total. For example, suppose you view a two-story building with a second floor that is smaller than the first floor. Obviously the area of the second floor's roof must be included in the total, but the visible surface area of the first floor's roof will not include the area directly underneath the second floor's roof.

Input

The input consists of several test cases corresponding to roof descriptions. A test case begins with a line containing integers V ($1 \le V \le 300$), which is the number of vertices, and T ($1 \le T \le 1000$), which is the number of triangles. Each of the next V lines contains the coordinates x, y, z of a vertex. The test case concludes with T triangle descriptions, each consisting of the three vertex indices describing the vertices of a roof triangle. Vertices are numbered sequentially in the order they appear in the input, starting with 1.

All coordinates are positive integers no greater than 100. No roof triangles are degenerate, and all pairs of roof triangles have disjoint interiors – that is, they may touch but they will not overlap or intersect.

The last test case is followed by a line containing two zeroes.

Output

For each test case, print a line containing the test case number (beginning with 1) followed by the total visible surface area of the roof, accurate to two fractional digits. Use the sample output format and print a blank line after each case.

Sample Input	Output for the Sample Input
6 2 10 10 10 10 20 10 20 10 10 10 10 20 10 20 20 20 20 20 1 2 3 4 5 6 3 1 10 10 10 10 18 16 20 10 10 1 2 3 0 0	Case 1: 75.00 Case 2: 50.00

This page intentionally left blank.

Problem I

Problem: Water Tanks

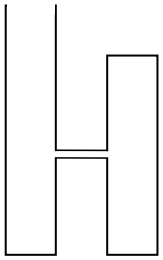
Input: tanks.in

The Aqua Container Management company manages water storage facilities. They are considering a system of storing water in a series of connected vertical tanks. Each tank has a horizontal cross-sectional area of one square meter, but the tanks have different heights. The base of each tank is at ground level. Each tank is connected by a pipe to the previous tank in the series and by another pipe to the next tank in the series. The pipes connecting the tanks are level and are at increasing heights (that is, the pipe connecting tank i to tank i+1 is at a higher level than the pipe connecting tank i to tank i-1.) Tank 1 is open so that air and water can flow into it freely at the top. All the other tanks are closed so that air and water can flow in and out only through the connecting pipes. The connecting pipes are large enough that water and air can flow through them freely and simultaneously but small enough that their dimensions can be ignored in this problem.

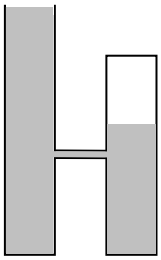
The series of tanks is filled by pouring water slowly into the top of tank 1, continuing until the water level reaches the top of tank 1. As the water level rises above the connecting pipes, water flows among the tanks.

Aqua Container Management needs a program to compute the cubic meters of water that can be poured into the series of tanks before the water level reaches the top of tank 1.

The figure below illustrates a simple case involving only two tanks. After the filling procedure is completed, the air in the upper part of the second tank is compressed (its air pressure is greater than one atmosphere), so the water level in the second tank is lower than the water level in the first tank.



Tanks before filling



Tanks after filling

The following physical principles are helpful in solving this problem (some of these are approximations that are acceptable for the purposes of this problem):

- Water flows downhill.
- In an open space, the air pressure is equal to one atmosphere.
- Air is compressible (the volume occupied by a given amount of air depends on pressure). Water is not compressible (the volume occupied by a given amount of water is constant, independent of pressure).
- Air pressure is the same everywhere within a closed space. If the volume of the closed space changes, the product of the volume and the air pressure within the space remains constant. For example, suppose an enclosed airspace has an initial volume V_1 and pressure P_1. If the volume of the airspace changes to V_2, then the new pressure P_2 satisfies $P_1V_1 = P_2V_2$.
- In a column of water below an airspace, the water pressure at a level D meters below the water surface is equal to the air pressure at the surface plus $0.097 \cdot D$ atmospheres. This is true regardless of whether the airspace is open or enclosed.
- In a connected body of water (for example, when two or more tanks are connected by pipes below the water line), the water pressure is constant at any given level.

Input

The input consists of several test cases representing different series of water tanks. Each test case has three lines of data. The first line contains an integer N ($2 \le N \le 10$) which is the number of tanks in the test case. The second line contains N positive floating point numbers that are the heights, in meters, of tanks 1 through N. The third line contains N-1 floating point numbers. On this line, the kth number represents the height above the ground of the pipe that connects tank k and tank k+1. The numbers on the third line are increasing (each number is greater than the preceding number).

The last test case is followed by a line containing the integer zero.

Output

For each test case, print a line containing the test case number (beginning with 1) followed by the amount of water, in cubic meters, that can be poured into tank 1 before the water level reaches the top of tank 1. Print the results with three digits to the right of the decimal point.

Print a blank line after the output for each test case. Use the format of the sample output.

Sample Input	Output for Sample Input
2 10.0 8.0 4.0 0	Case 1: 15.260

Problem J

Tunnels

Input File: tunnels.in

Curses! A handsome spy has somehow escaped from your elaborate deathtrap, overpowered your guards, and stolen your secret world domination plans. Now he is running loose in your volcano base, risking your entire evil operation. He must be stopped before he escapes!

Fortunately, you are watching the spy's progress from your secret control room, and you have planned for just such an eventuality. Your base consists of a complicated network of rooms connected by non-intersecting tunnels. Every room has a closed-circuit camera in it (allowing you to track the spy wherever he goes), and every tunnel has a small explosive charge in it, powerful enough to permanently collapse it. The spy is too quick to be caught in a collapse, so you'll have to strategically collapse tunnels to prevent him from traveling from his initial room to the outside of your base.

Damage to your base will be expensive to repair, so you'd like to ruin as few tunnels as possible. Find a strategy that minimizes the number of tunnels you'll need to collapse, no matter how clever the spy is. To be safe, you'll have to assume that the spy knows all about your tunnel system. Your main advantage is the fact that you can collapse tunnels whenever you like, based on your observations as the spy moves through the tunnels.

Input

The input consists of several test cases. Each test case begins with a line containing integers R ($1 \le R \le 50$) and T ($1 \le T \le 1000$), which are the number of rooms and tunnels in your base respectively. Rooms are numbered from 1 to R. T lines follow, each with two integers x, y ($0 \le x,y \le R$), which are the room numbers on either end of a tunnel; a 0 indicates that the tunnel connects to the outside. More than one tunnel may connect a pair of rooms.

The spy always starts out in room 1. Input is terminated by a line containing two zeros.

Output

For each test case, print a line containing the test case number (beginning with 1) followed by the minimum number of tunnels that must be collapsed, in the worst case. Use the sample output format and print a blank line after each test case.

Sample Input	Output for the Sample Input
4 6 1 2 1 3 2 4 3 4 4 0 4 0 4 6 1 2 1 3 1 4 2 0 3 0 4 0 0 0	Case 1: 2 Case 2: 2

This page intentionally left blank.