A - Winter Roads

Source file name: `winter.c`, `winter.cpp` *or* `winter.java`

Winter is coming, and the citizens of Winterfell are making preparations for the long months ahead. One particular concern is the stability of the roads and bridges; supply lines must stay uninterrupted as the season progresses.

The civil engineers of Winterfell would like to model possible scenarios of roads failing and repairs that need to be made. In their model, each road connects two landmarks in the city, and each road has a certain carrying capacity. Trucks travel from landmark to landmark, and can only travel on sufficiently sturdy roads. In particular, a supply truck with weight w can travel on a road with capacity c iff $w \leq c$. Whether by nature or by accident, the capacity of a road can decrease over time. In addition, repairs can increase the carrying capacity of a road. While roads change, however, supply trucks still need to be able to get from source to destination, and the engineers would like to test whether trucks can still make certain runs between landmarks.

Given a series of changes to the roads and some supply truck runs, help the engineers check whether or not supplies can still be delivered as winter strikes.

Input

There will be several test cases in the input. Each test case will begin with a line with two integers n, m ($1 \leq n \leq 1,000$, $1 \leq m \leq 100,000$), where n is the number of landmarks, and m is the number of roads between them. This will be followed by m lines, each with three integers a, b, and c ($1 \leq a, b \leq n$ and $1 \leq c \leq 1,000,000,000$) representing a road between landmarks a and b with carrying capacity c. Roads are numbered 1, 2, 3, ..., m in the order that they appear in the input.

Next will be a line with a single integer e ($1 \leq e \leq 100,000$), indicating the number of events. This will be followed by e lines, consisting of a capital letter followed by integers:

B r c: Indicates that road r breaks down. Its carrying capacity decreases to c. ($1 \leq r \leq m$, $1 \leq c < 1,000,000,000$)

R r c: Indicates that road r is repaired. Its carrying capacity increases to c. ($1 \leq r \leq m$, $1 < c \leq 1,000,000,000$)

S a b w: Check whether or not a supply truck with weight w can travel from landmark a to landmark b. ($1 \leq a, b \leq n$, $1 \leq w \leq 1,000,000,000$)

Only capital letters **B**, **R** or **S** will appear. All events occur in the order given in the input. There will be at most 2,000 breakdown / repair events in the input. The input will end with a line with two 0s.

The input must be read from standard input.

Output

For each **S** *a b w* query, in order, output 1 if the there is a path from a to b that can support a truck of weight w, and 0 if there is not. Output each number on its own line, with no spaces. Do not print any blank lines between outputs.

The output must be written to standard output.

Sample Input	**Sample Output**
3 4 1 2 3 2 3 3 2 1 1 1 2 1 6 S 1 2 4 S 2 3 2 R 1 4 S 1 2 4 B 2 1 S 2 3 2 0 0	0 1 1 0

B - Can of Worms

Source file name: `worms.c`, `worms.cpp` *or* `worms.java`

There is an old adage about opening a can of worms. A lesser known adage is one about shooting a can of exploding worms with a BB gun.

Imagine we place some cans of exploding worms on a long, straight fence. When a can is shot, all of the worms inside will explode. Different types of worms have different blast radii. Each can contains only one kind of worm.

When a can explodes, if another can is in the blast radius, then that can will also explode, possibly creating a chain reaction. Each can explodes only once. This process continues until all explosions stop.

For each can, suppose that it is the only can shot. How many cans in total will explode?

Input

There will be several test cases in the input. Each test case will begin with a line with a single integer n ($1 \leq n \leq 100,000$) representing the number of cans on that fence. Each of the next n lines will have two integers x ($-10^9 \leq x \leq 10^9$) and r ($1 \leq r \leq 10^9$), where x is the location of the can on the fence and r is the blast radius. No two cans will occupy the same location. The input will end with a line with a single 0.

The input must be read from standard input.

Output

For each fence, print n integers on a single line separated by single spaces. The ith integer represents the number of cans that will explode if the ith can is the one that is shot. Output no extra spaces, and do not print any blank lines between outputs.

The output must be written to standard output.

Sample input	**Sample output**
3 4 3 -10 9 -2 3 12 2 2 7 7 10 1 19 3 23 12 29 8 33 1 35 17 39 2 40 1 46 11 52 3 0	1 2 1 1 3 1 1 9 9 1 9 2 2 9 1

C - Automatic Trading

Source file name: `trading.c`, `trading.cpp` *or* `trading.java`

A brokerage firm is interested in detecting automatic trading. They believe that a particular algorithm repeats itself; that is, it makes the same sequence of trades at a later time. The firm has identified a set of 26 key stocks that they believe are likely to be traded in concert, and they've encoded a series of trades as a string of letters: the letter itself indicates the stock, upper case indicates a *buy*, lower case indicates a *sell*. They want you to write a program to determine, for any two starting points, the longest sequence of identical trades from those two starting points, starting from and including those starting points as the first trade in the sequence.

Input

There will be several test cases in the input. Each test case will start with a string s on the first line, consisting solely of upper case and lower case letters ($1 \leq length(s) \leq 100,000$). On the next line will be an integer q ($1 \leq q \leq 100,000$), indicating the number of queries. The following q lines each describe a query with two integers, i and j ($0 \leq i < j < length(s)$), which represent two zero-based positions in the string. The input will end with a line containing only an asterisk ('*').

The input must be read from standard input.

Output

For each query, output a single integer, indicating the length of the longest sequence of trades starting at i which is identical to the sequence of trades starting at j. Output no spaces. Do not print any blank lines between lines of output.

The output must be written to standard output.

Sample Input	**Sample Output**

`ABABABcABABAbAbab`
`3`
`0 2`
`1 6`
`0 7`
`SheSellsSeashellsByTheSeaShore`
`4`
`8 22`
`1 20`
`8 25`
`0 1`
`*`
	`4`
	`0`
	`5`
	`3`
	`4`
	`1`
	`0`

D - 3D Printer

Source file name: `printer.c`, `printer.cpp` *or* `printer.java`

3D printing is a technique for manufacturing items from a digital template. The printer lays down layers of a polymer material, building an entire 3D object as a series of flat plates of varying shapes, stacked upon one another. The polymer is initially sticky enough so that the plates printed on top of one another will adhere. After the object dries or cures, the resulting objects can be quite durable.

Consider a 3D printer in which objects to be printed are described as a template, consisting of a combination of several convex polyhedra (i.e., flat-surfaced objects such that a line from one interior point to another interior point never passes outside the volume of the object). Write a program to determine the total volume of polymer required to sculpt an object from a given template.

Input

There will be several test cases in the input. Each test case will begin with a line with a single integer n ($1 \leq n \leq 100$) representing the number of polyhedra in that template.

The subsequent lines describe the n polyhedra. Each polyhedron begins with a line containing an integer f ($3 < f < 30$), which is the number of faces on the polyhedron. Following that line is a series of lines describing polygons that comprise the faces. Each such line begins with an integer v ($3 \leq v \leq 24$), which is the number of vertices. Following v on the same line will be $3 * v$ real numbers, representing the v vertices as (x, y, z) coordinates. For example, if $v = 3$, then the line would be

v $x1$ $y1$ $z1$ $x2$ $y2$ $z2$ $x3$ $y3$ $z3$

All coordinates will be in the range [-100..100]. Vertices are presented in sequential order; there will be an edge of the polygon from ($x1$, $y1$, $z1$) to ($x2$, $y2$, $z2$), from ($x2$, $y2$, $z2$) to ($x3$, $y3$, $z3$), and so on. The polygons are closed, so there is an implied edge from the last vertex in a polygon back to the first. All of the vertices of a face will be coplanar. Edges will not cross, and each vertex will lie on exactly two edges. No three (or more) vertices in a polygon will be collinear.

None of the polyhedra in any given test case will overlap. The input will end with a line with a single 0.

The input must be read from standard input.

Output

For each template, print a real number on its own line, indicating the volume of polymer required in cubic centimeters. The volume should be printed with a precision of two decimal

places, rounded. Do not print any spaces. Do not print any blank lines between answers.

The output must be written to standard output.

Sample Input	**Sample output**
2 6 4 10 10 0 10 15 0 15 15 0 15 10 0 4 10 10 0 10 15 0 10 15 20 10 10 20 4 10 15 0 15 15 0 15 15 20 10 15 20 4 15 15 0 15 10 0 15 10 20 15 15 20 4 10 10 0 15 10 0 15 10 20 10 10 20 4 10 10 20 10 15 20 15 15 20 15 10 20 6 4 0 0 0 0 25 0 25 25 0 25 0 0 4 0 0 0 0 25 0 0 25 0.5 0 0 0.5 4 0 25 0 25 25 0 25 25 0.5 0 25 0.5 4 25 25 0 25 0 0 25 0 0.5 25 25 0.5 4 25 0 0 0 0 0 0 0 0.5 25 0 0.5 4 0 0 0.5 0 25 0.5 25 25 0.5 25 0 0.5 0	812.50

E - Flooding Fields

Source file name: `flooding.c, flooding.cpp` *or* `flooding.java`

Farmer John has a problem: it's raining, and his pastures are at risk of flooding. Luckily he's invested in a state-of-the-art drain system that ensures that the water level is the same across the farm. He hasn't been as lucky with the terrain. Farmer John's cows can only stand on dry land: if the land becomes wet, any cow on that land drowns. Luckily, cows can move one grid sector each hour, giving them a chance to escape the water (the levels of which rise and fall each hour).

Farmer John has divided his field into grid sectors, which he indexes with a row and column. Each grid sector is small enough to only be able to hold one cow.

Given a description of Farmer John's field, the starting locations of his cows, and the height of the water at each hour, and assuming that Farmer John's cows are extremely intelligent and prescient, and so can always make the best move, what is the maximum number of Farmer John's cows that could survive?

Input

There will be several test cases in the input. Each test case will begin with a line with three integers, n ($1 \leq n \leq 100$), k ($0 \leq k \leq 100$), and h ($1 \leq h \leq 24$), where n represents the size of Farmer John's field (it's $n \times n$ grid sectors), k is the number of cows in the field, and h is the number of hours he needs to track.

Each of the next n lines will contain n integers each, which represent the height of each grid sector of Farmer John's field ($0 \leq height \leq 100$). The first row in the input is row 0, and the last is row $n - 1$. The first column in each row is column 0, and the last is column $n - 1$.

The following k lines will each contain a pair of integers, r followed by $c (0 \leq r, c < n)$. Each line indicates the grid sector position of one cow at hour 0, where r is the row and c is the column. No two cows will be in the same position.

The next h lines will each contain a single integer, indicating the level of the flood at that hour ($0 \leq level \leq 100$). These lines are given in order: hour 1 first, then hour 2, and so on. Note that the times start at hour 1, and the cows' positions are given at hour 0, so the cows have an opportunity to move (or MOOOOve) before the first hour's flood. A grid square is considered to be flooded if its $height \leq level$ of the flood at a given hour.

The input will end with a line with three 0s.

The input must be read from standard input.

Output

For each test case, output a single integer denoting the maximum possible number of surviving cows. Do not output any spaces, and do not print any blank lines between answers.

The output must be written to standard output.

Sample input	**Sample output**
3 1 3 2 1 2 3 1 3 2 4 2 1 1 0 2 1 0 0 0	1

F - Goat Ropes

Source file name: `goat.c`, `goat.cpp` *or* `goat.java`

A farmer has n goats. Coincidentally, he also has n fixed posts in a field where he wants the goats to graze. He wants to tie each goat to a post with a length of rope. He wants to give each goat as much leeway as possible - but, goat ropes are notorious for getting tangled, so he cannot allow any goat to be able to wander into another goat's territory. What is the maximum amount of rope he could possibly use?

Input

There will be multiple test cases in the input. Each test case will begin with an integer n ($2 \leq n \leq 50$), indicating the number of posts in the field. On each of the next n lines will be a pair of integers, x and y ($0 \leq x \leq 1,000$, $0 \leq y \leq 1,000$) which indicate the cartesian coordinates (in meters) of that post in the field. No two posts will be in the same position. You may assume that the field is large enough that the goats will never encounter its border. The input will end with a line with a single 0.

The input must be read from standard input.

Output

For each test case, output a single floating point number, which indicates the maximum amount of rope that the farmer could possibly use, in meters. Output this value with exactly two decimal places, rounded. Output no spaces, and do not output a blank line between answers.

The output must be written to standard output.

Sample input	**Sample output**
2 250 250 250 750 3 250 250 500 500 250 750 0	500.00 603.55

G - Job Postings

Source file name: `job.c`, `job.cpp` *or* `job.java`

Your university has jobs available for students, and the administration needs for you to help them assign students to jobs. The goal is to have students select their desired positions, and then allocate each student to one of the positions so that the maximum satisfaction is achieved.

Each student selects four positions, in the order of desirability. First position is most wanted, next position is next most wanted, if the first position is not available for this student, and so on.

Students have seniority, based their year of study. Third-year students' selections should have more weight than first-year students' selections.

The administration wants for you to use the following satisfaction matrix:

Job/Position choice:	1st	2nd	3rd	4th
1st year student:	4	3	2	1
2nd year student:	8	7	6	5
3rd year student:	12	11	10	9

Your task is to assign students to positions in a way that maximizes the sum of all students' satisfaction according to the above matrix. Each student must get a position, but all positions may not be filled.

Input

There will be multiple test cases in the input. Each test case will begin with two integers, n ($4 \leq n \leq 140$) and m ($1 \leq m \leq 70$), where n is the number of postings and m is the number of students. Each of the next n lines will contain a single integer p ($1 \leq p \leq 10$), indicating the number of positions available for that job posting. The job postings are listed in order, from job 0 to job $n - 1$.

Following the job postings will be m lines describing the students. Each student line will have five integers:

y $c1$ $c2$ $c3$ $c4$

Where y (y=1, 2 or 3) is the student's year of study, and $c1$, $c2$, $c3$ and $c4$($0 \leq c1, c2, c3, c4 < n$, all four unique) indicating the student's choice of job postings, in order of preference.

It is guaranteed for every test case that a solution exists where every student can get one of the positions on their choice list.

The input will end with a line with two 0s.

The input must be read from standard input.

Output

For each test case, output a single integer, which indicates the maximum satisfaction achievable. Output no spaces, and do not output a blank line between answers.

The output must be written to standard output.

Sample input	**Sample output**
4 4 1 1 1 1 1 0 1 2 3 2 0 1 2 3 3 0 1 2 3 3 0 1 2 3 4 4 4 4 4 4 1 0 1 2 3 2 0 1 2 3 3 0 1 2 3 3 0 1 2 3 0 0	30 36

H - Overlapping Maps

Source file name: `maps.c`, `maps.cpp` *or* `maps.java`

Fred and Sam are traveling together. Both have maps of the area. The maps cover exactly the same territory, and have exactly the same ratio of width to height, but Sam's is at a smaller scale than Fred's, so it's a bit smaller. Fred puts his map on a table. Sam throws his map on top of it. The smaller map is offset and rotated, but it still fits entirely on top of the bigger map. Fred sticks a pin in the smaller map. The pin goes all the way through to the bigger map. Sam is amazed! He says "Wow! Do you realize that the position of that pin represents the same place on BOTH maps?" Fred says "it's really not so amazing. There has to be one such point!"

Given the dimensions of a large map, and the offset, scale and rotation of a smaller map that is entirely on top of the larger map, find the single point that represents the same place on both maps.

Input

There will be several test cases in the input. Each test case will consist of a single line with six integers:

w h x y s r

The first two integers w and h ($0 < w, h \leq 1,000$) are the width and height of the larger map. The larger map will be on a plane, with the southwest corner at the origin, the northwest corner at (0,h), the southeast corner at (w,0), and the northeast corner at (w,h).

The next two integers, x and y ($0 \leq x \leq w$, $0 \leq y \leq h$), represent the (x, y) coordinate on the plane of the southwest corner of the smaller map.

The integer s ($0 < s < 100$) represents the scale of smaller map as a percentage of the larger map ($s = 50$ means that the smaller map has half the width and half the height of the larger map).

The integer r ($0 \leq r < 360$) is the angle, in degrees, of counter-clockwise rotation of the smaller map around its southwest corner ($r = 90$ means that the southeast corner is rotated to be due north of the southwest corner).

The smaller map is guaranteed to lie completely within the borders of the larger map. The input will end with a line with six 0s.

The input must be read from standard input.

Output

For each test case, output two real numbers, x and y, representing the (x, y) coordinate of the point where both maps represent the same place. Output the numbers to two decimal places of accuracy, with a single space between them. Do not output any extra spaces. Do not put blank lines between answers for different cases.

The output must be written to standard output.

Sample input	**Sample output**
100 100 50 50 25 0 100 100 50 50 25 45 0 0 0 0 0 0	66.67 66.67 45.59 70.53

I - Unreal State

Source file name: `unreal.c, unreal.cpp` *or* `unreal.java`

A dishonest landowner is selling off plots of land. He's selling it in large, rectangular plots, but many of the plots overlap, so he's actually selling the same land multiple times! It's not real estate, it's unreal estate!

Given a description of the possibly overlapping rectangular plots of land that the dishonest landowner has sold, determine the total actual area of land covered by them.

Input

There will be several test cases in the input. Each test case will begin with a line with a single integer n ($0 < n \leq 5,000$), indicating the number of plots of land sold. The next n lines will each have a description of a rectangular plot of land, consisting of four real numbers:

$x1$ $y1$ $x2$ $y2$

where ($x1$, $y1$) is the southwest corner, and ($x2$, $y2$) is the northeast corner ($-1,000 \leq x1 < x2 \leq 1,000$, $-1,000 \leq y1 < y2 \leq 1,000$). Every plot will have an area of at least 1.

The input will end with a line with a single 0.

The input must be read from standard input.

Output

For each test case, output a single real number, which represents the total actual area covered by all of the rectangular plots of land. Output this number with exactly two decimal places, rounded. Do not print any spaces. Do not print any blank lines between outputs.

The output must be written to standard output.

Sample Input	**Sample output**
2 0 0 100 100 30 30 60 60 2 0 100 300 200 100 0 200 300 0	10000.00 50000.00

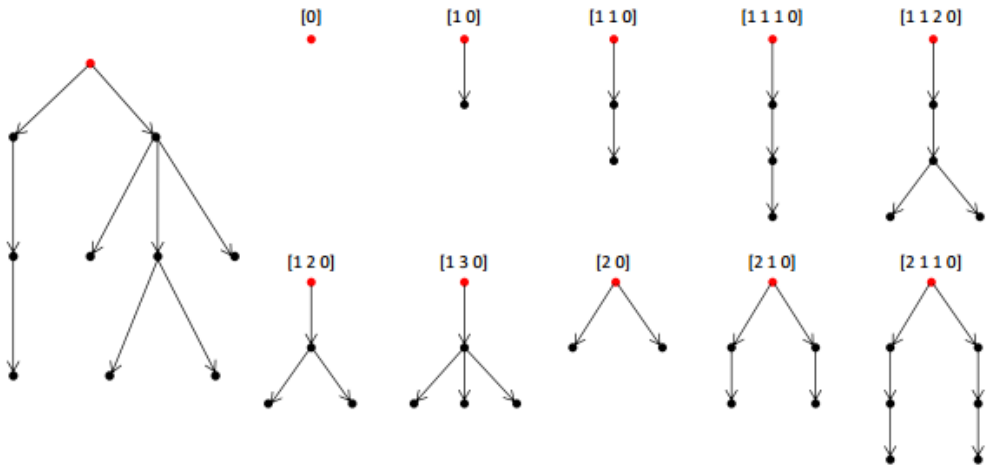
J - Uniform Subtrees

Source file name: `subtrees.c`, `subtrees.cpp` *or* `subtrees.java`

Define a Uniform tree as one in which all of the nodes at a given level (i.e. distance from the root) have the same degree (i.e. number of children). Since all of the nodes at a given level have the same number of children, a uniform tree can be represented as simply a list of integers, indicating the number of children at each level. For example, the list [2 3 5 0] represents a tree where the root has 2 children, each child of the root has 3 children, each grandchild has 5 children, and each great-grandchild has no children, and is therefore a leaf.

For the purposes of this problem, redefine Subtree as a connected subgraph of a tree that includes the tree's root. This is a bit different than the typical definition of Subtree.

Given a description of a tree, find all of the unique Uniform Subtrees of that tree. For example, here is a tree and all of its unique uniform subtrees:



Input

There will be several test cases in the input. Each test case will will consist of a single tree, represented as a single string on one line. The string will be a sequence of matched opening and closing parentheses. Each matched pair represents a node, and the string between represents its children. There will not be more than 4,000 nodes in the tree. There will be no whitespace, or any other characters, in the string. The input will end with a line with a single 0.

The input must be read from standard input.

Output

For each test case, output every unique uniform subtree of the given tree as a list of integers, one subtree (and thus one list) per line. Print a single space between integers, and no spaces

anywhere else. Do not print any blank lines between lists, or between test cases. Print the lists for a given test case sorted by the first element, then the second, then the third, and so on.

The output must be written to standard output.

Sample input	**Sample output**
((((()))(()(()())())))	0
(())	1 0
0	1 1 0
	1 1 1 0
	1 1 2 0
	1 2 0
	1 3 0
	2 0
	2 1 0
	2 1 1 0
	0
	1 0