WEEK0 - ASSIGNMENT 5

MS Connect

Wipro Technologies

Objectives

- Get comfortable with .NET.
- Start thinking more carefully.
- Solve some problems in C#

Reasonable

- Communicating with colleagues about problem problems in English (or some other spoken language).
- Discussing the assignment material with others in order to understand it better.
- Helping a colleagues identify a bug in his or her code, as by viewing, compiling, or running his or her code, even on your own computer.
- Incorporating snippets of code that you find online or elsewhere into your own code, provided that those snippets are not themselves solutions to assigned problems and that you cite the snippets' origins.
- Sending or showing code that you've written to someone, possibly a colleagues, so that he or she might help you identify and fix a bug.

Deliverables

- Estimate the required time to solve the problem.
- Identify & write test cases (in excel) before you start coding.
- Draw the flow chart/block diagrams for each solutions.
- Document the solution approach in detail for your case/problem.
- Document your solutions in word file.
- Write proper comments for each line in your source code.
- Document the output of your program.
- Your program should address the problem, it should give accurate results.

1. Rain

In recent years, hurricanes and tsunamis have shown the destructive power of water. That destructive power is not restricted to the sea, however. Heavy rain may cause floods, destroying people's houses and fields. Using intricate models, scientists try to predict where the water will accumulate as a result of significant rain.

One of the ways to model a hilly landscape is triangulation, which approximates a surface using triangles. Triangle sides are entirely shared between two adjacent triangles, with the exception of triangles at the region boundary, which may have some sides not shared.

Imagine you have a triangulation model of a landscape. Now the rain starts pouring down – some water flows to the sea, and the rest gets trapped by the landscape to form lakes. Your task is to write a program that determines how many lakes are formed and the water level in each of them. Assume that the rain is heavy enough to fill all lakes up to their maximal levels.

For any lake, it is possible to sail between any two points on its surface (except the boundaries) with a boat whose size is arbitrarily small, but not zero. Therefore, if two lakes share only points (or one point) having a zero depth, they are considered different lakes.

Input

The input contains several test cases. Each test case starts with a line containing two integers, $p \geq 3$, which is the number of points, and $s \geq 3$, which is the number of sides of the triangulation. Each of the next ***p*** lines describes a triangulation point.

The description of each point starts with a two-letter code that is unique for the test case. The code is followed by three integers that describe the point in the format *x y h*. Here x and y $(-10000 \leq x,y \leq 10000)$ are two dimensional coordinates and h $(0 \leq h \leq 8848)$ is the height of the point above sea level.

Each of the next s lines contains the description of one side of a triangle. The description of a side consists of two different two-letter codes that specify the endpoints of the side. The projection of the lines to the *xy-plane* satisfies the following conditions:

- No side intersects any other side except at its endpoints.
- The points and sides together form a triangulation of a single connected region.
- There are no "holes" inside the region (that is, the boundary forms a single closed polygonal curve).

You may consider all points outside the triangulated region to have lower heights than the closest point of the region boundary. In other words, if the water gets to a boundary of the region, it flows out freely.

The last line of the input contains two zeroes.

Output

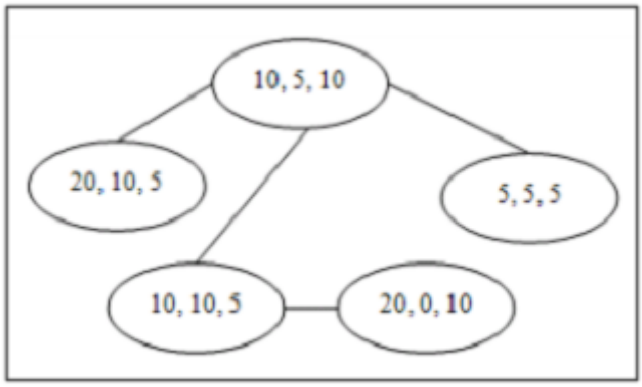
For each test case, display the case number and the levels of all different lakes located inside the given region, each on a separate line. The levels are heights above sea level and should be printed in non-decreasing order. If no lakes are formed display a single 0. Follow the format of the sample output.

Sample Input	Output for the Sample Input
3 3 AA 0 0 0 BB 0 1 0 CC 1 0 0 AA BB AA CC BB CC 7 12 aa 1 1 5 bb 1 3 5 xX 0 2 10 XY 0 4 15 XZ 3 4 11 xy 0 0 15 xz 3 0 15 xX XZ XY XZ xX XY xX xy xX xz xz xy aa xX aa xy aa xz bb xX bb XY bb XZ 0 0	Case 1: 0 Case 2: 10 10

2. Castle

Wars have played a significant role in world history. Unlike modern wars, armies in the middle ages were principally concerned with capturing and holding castles, the private fortified residences of lords and nobles. The size of the attacking army was an important factor in an army's ability to capture and hold one of these architectural masterpieces.

A certain minimum number of soldiers were required to capture a castle. Some soldiers were expected to die during the attack. After capturing the castle, some soldiers were required to remain in the castle to defend it against attacks from another enemy. Of course, those numbers were different for different castles. Commanders of the armies were obliged to consider the number of soldiers required for victory.



For example, there are five castles in the region map shown in above Figure. The castle at the lower right requires at least 20 soldiers to wage a winning attack. None are expected to perish during the attack, and 10 soldiers must be left in the castle when the army moves on.

In this problem you must determine the minimum size of an army needed to capture and hold all the castles in a particular region. For reasons of security, there is exactly one (bi-directional) route between any pair of castles in the region. Moving into the neighborhood of an uncaptured castle begins an attack on that castle. Any castle can serve as the first castle to be attacked, without regard for how the army got there. Once any castle has been captured, the requisite number of soldiers is left in the castle to defend it, and the remainder of the army moves on to do battle at another castle, if any remain uncaptured. The army may safely pass through the neighborhood of a castle that it has already captured. But because of the potential for attacks, the army may traverse the route between a pair of castles no more than twice (that is, at most once in each direction).

Input

The input contains multiple test cases corresponding to different regions. The description of the castles in each region occupies several lines. The first line contains an integer n ≤ 100 that is the number of castles in the region. Each of the next n lines contains three integers a, m, and g (1 ≤ a ≤ 1000, 0 ≤ m ≤ a, 1 ≤ g ≤ 1000), that give the minimum number of soldiers required to successfully attack and capture a particular castle, the number of soldiers that are expected to die during the attack, and the number of soldiers that must be left at the castle to defend it. The castles are numbered 1 to n, and the input lines describing them are given in increasing order of castle numbers. Each of the remaining n – 1 lines in a test case has two integers that specify the castle numbers of a pair of castles that are connected by a direct route.

A line containing 0 follows the description of the last region.

Output

For each test case, display the case number and the minimum number of soldiers in the army needed to conquer all the castles in the region. Follow the format shown in the sample output.

Sample Input	Output for the Sample Input
3 5 5 5 10 5 5 5 1 1 1 3 2 3 5 10 5 10 20 10 5 10 10 5 5 5 5 20 0 10 1 2 1 3 1 4 3 5 0	Case 1: 22 Case 2: 65

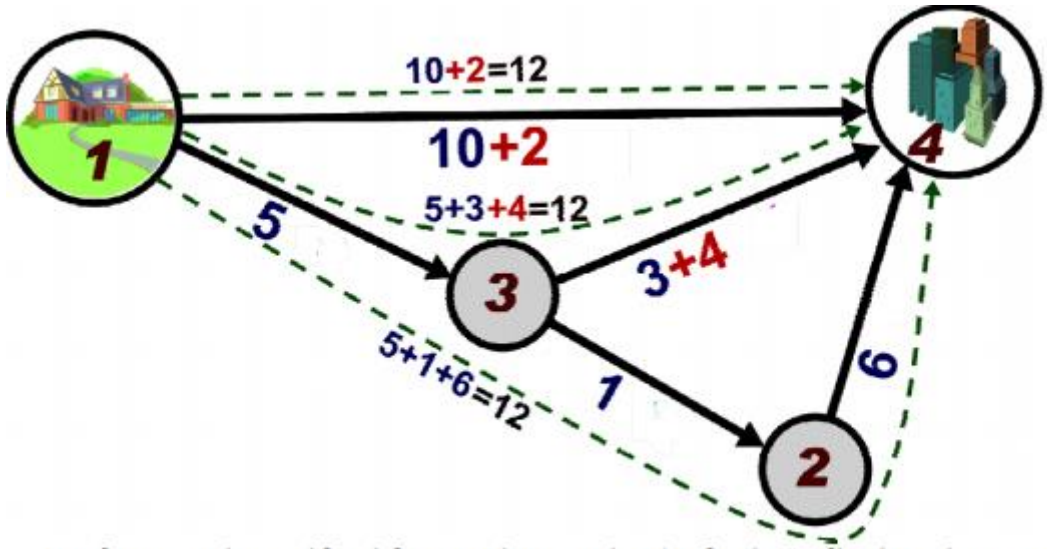
3. Traffic

Handling traffic congestion is a difficult challenge for young urban planners. Millions of drivers, each with different goals and each making independent choices, combine to form a complex system with sometimes predictable, sometimes chaotic behavior. As a devoted civil servant, you have been tasked with optimizing rush-hour traffic over collections of roads.

All the roads lie between a residential area and a downtown business district. In the morning, each person living in the residential area drives a route to the business district. The morning commuter traffic on any particular road travels in only one direction, and no route has cycles (morning drivers do not backtrack).

Each road takes a certain time to drive, so some routes are faster than others. Drivers are much more likely to choose the faster routes, leading to congestion on those roads. In order to balance the traffic as much as possible, you are to add tolls to some roads so that the perceived "cost" of every route ends up the same. However, to avoid annoying drivers too much, you must not levy a toll on any driver twice, no matter which route he or she takes.

Below figure shows a collection of five roads that form routes from the residential area (at intersection 1) to the downtown business district (at intersection 4). The driving cost of each road is written in large blue font. The dotted arrows show the three possible routes from 1 to 4. Initially the costs of the routes are 10, 8 and 12. After adding a toll of cost 2 to the road connecting 1 and 4 and a toll of cost 4 to the road connecting 3 and 4, the cost of each route becomes 12.



Roads connecting residential area at intersection 1 to business district at intersection 4

You must determine which roads should have tolls and how much each toll should be so that every route from start to finish has the same cost (driving time cost + possible toll) and no route contains more than one toll road.

Additionally, the tolls should be chosen so as to minimize the final cost. In some settings, it might be impossible to impose tolls that satisfy the above conditions.

Input

Input consists of several test cases. A test case starts with a line containing an integer N ($2 \le N \le 50000$), which is the number of road intersections, and R ($1 \le R \le 50000$), which is the number of roads. Each of the next R lines contains three integers xi, yi, and ci ($1 \le xi, yi \le N$, $1 \le ci \le 1000$), indicating that morning traffic takes road i from intersection xi to intersection yi with a base driving time cost of ci. Intersection 1 is the starting residential area, and intersection N is the goal business district. Roads are numbered from 1 to R in the given input order.

Every intersection is part of a route from 1 to N, and there are no cycles. The last test case is followed by a line containing two zeros.

Output

For each test case, print one line containing the case number (starting with 1), the number of roads to toll (T), and the final cost of every route. On the next T lines, print the road number *i* and the positive cost of the toll to apply to that road. If there are multiple minimal cost solutions, any will do. If there are none, print ***No solution.*** Follow the format of the sample output.

Sample Input	**Output for Sample Input**
4 5 1 3 5 3 2 1 2 4 6 1 4 10 3 4 3 3 4 1 2 1 1 2 2 2 3 1 2 3 2 0 0	Case 1: 2 12 4 2 5 4 Case 2: No solution

4. Conveyor Belt

Many mechanical systems work with rotating shafts connected with conveyor belts. The shafts have a variety of sizes and rotate in either a clockwise or a counterclockwise manner. The exact way in which a belt will connect two shafts depends on their rotations, as shown in Figures 1 and 2.

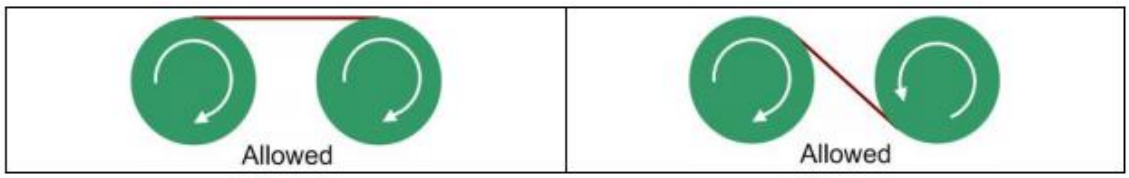


Figure 1

Figure 2

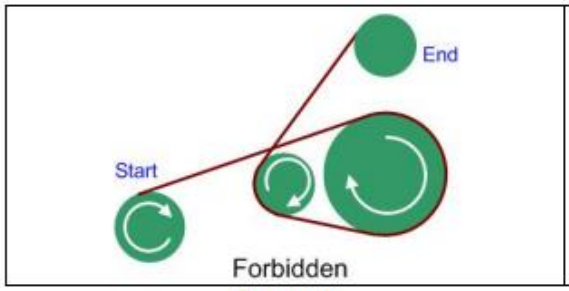


Figure 3

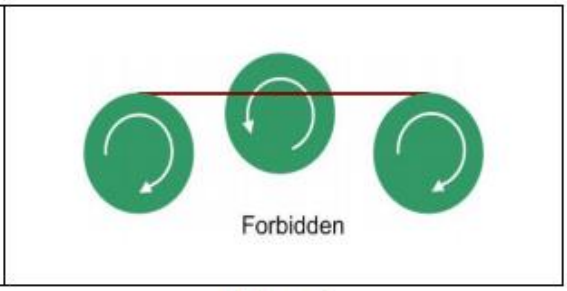


Figure 4

One task in setting up such mechanical systems is to link together two given shafts, subject to these constraints:

• If the two shafts being connected are too far apart, the belt may start to vibrate chaotically when perturbed slightly. To prevent this, you can connect shafts only when the distance between the points where the belt leaves one shaft and touches the other is *less than* some distance *d* (the exact value of *d* varies depending on the type of belt).

• No belt can cross over itself, as shown in Figure 3.

• No belt can pass through another shaft (or touch one rotating the wrong way), as shown in Figure 4.

• The belt is not a loop; it goes in only one direction, from the starting shaft to the ending shaft. The starting shaft "pushes" the belt and the ending shaft "pulls" it.

As an example, consider the problem of connecting shaft A to shaft D in Figure 5. Suppose that the distance needed to connect A to C (shown in blue dashed line) or to connect B to D is greater than the limit allowed. Then the shortest distance to connect A to D is shown in solid line, going from shaft A to B to C and then D. Notice that the connection cannot go from B to E and then D as the belt would cross itself.

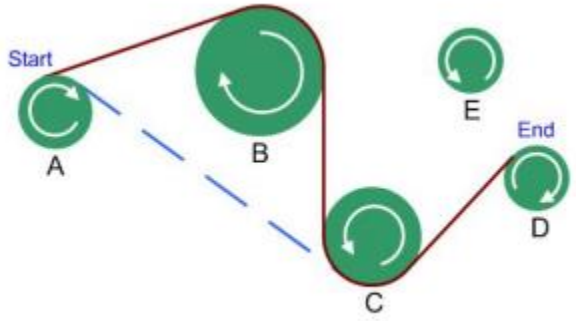


Figure 5

You must write a program that calculates the minimum length of the belt to connect the two given shafts.

Input

The input consists of multiple test cases. Each test case starts with a line containing an integer N ($1 \le N \le 20$) indicating the number of shafts, numbered from 0 to *N -1*. Starting on the next line are *N* four-tuples of the form *x y r s*, where x and y are the integer coordinates of a shaft center, *r* is the integer radius of the shaft, and s is either "C" for clockwise or "CC" for counterclockwise ($0 \le x, y \le 10000$ and $0 < r \le 1000$). Positive *x* is right and positive *y* is up. The first four-tuple specifies shaft 0, the second one shaft 1 and so on. These four-tuples may extend over multiple lines, though no four-tuple will be split across two lines. No two shafts touch or overlap. The last line of each test case contains two integers i and j indicating the starting and ending shafts, followed by a floating point value *d* specifying the maximum distance constraint.

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

Output

For each test case, print the case number (starting with 1) followed by the length of the path of minimum distance. Print ***Cannot reach destination shaft*** if the destination shaft cannot be reached. Use the format shown in the sample output.

When measuring the distance, start where the belt first leaves the starting shaft and end where the belt first touches the ending shaft. Your distance calculation must include both the length of belt between shafts as well as the distance the belt travels around intermediate shafts. Your answer should be rounded to the nearest hundredth, though you need not print trailing zeroes after the decimal point.

Sample Input	Output for the Sample Input
5 24 50 14 C 93 78 20 C 118 8 15 CC 167 32 13 C 159 88 15 CC 0 3 82.5 5 24 50 14 C 93 78 20 C 118 8 15 CC 167 32 13 C 159 88 15 C 0 3 82.5 5 24 50 14 C 93 78 20 C 118 8 15 CC 167 32 13 C 159 88 15 C 0 3 8.5 0	Case 1: length = 271 Case 2: length = 228.23 Case 3: Cannot reach destination shaft