Colombian Collegiate Programming League

CCPL 2018

Round 7 – June 9

Problems

This set contains 10 problems; pages 1 to 18.

(Borrowed from several sources online.)

Official site `http://programmingleague.org`

Follow us on Twitter `@CCPL2003`

A - Abstract Art

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

Arty has been an abstract artist since childhood, and his works have taken on many forms. His latest (and most pricey) creations are lovingly referred to as Abstract Art within the abstract art community (they're not the most original bunch when it comes to loving nicknames). Here's an example of one of Arty's recent works:

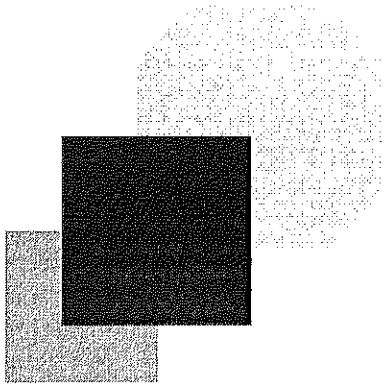

Figure 1: An example of Arty's art.

As you can see, Abstract Art is created by painting (possibly overlapping) polygons. When Arty paints one of his designs he always paints each polygon completely before moving on to the next one.

The price of individual pieces of Arty's Abstract Art varies greatly based on their aesthetic appeal, but collectors demand two pieces of information about each painting:

1. the total amount of paint used, and
2. the total amount of canvas covered.

Note that the first value will be larger than the second whenever there is overlap between two or more polygons. Both of these values can be calculated from a list containing the vertices of all the polygons used in the painting, but Arty can't waste his time on such plebeian pursuits – he has great art to produce! I guess it's left up to you.

Input

The first line of input contains a single integer n ($1 \leq n \leq 100$) representing the number of polygons to be painted. Following this are n lines each describing a painted polygon. Each polygon description starts with an integer m ($3 \leq m \leq 20$) indicating the number of sides in the polygon, followed by m pairs of integers x y ($0 \leq x, y \leq 1000$) specifying the coordinates of the vertices of the polygon in consecutive order. Polygons may be concave but no polygon will cross itself. No point on the canvas will be touched by more than two polygon border segments.

The input must be read from standard input.

Output

Display both the total amount of paint used and the amount of canvas covered. Output these numbers with exactly three decimal places, rounded.

The output must be written to standard output.

Sample Input	Sample Output
3 8 7 10 7 17 10 20 17 20 20 17 20 10 17 7 10 7 4 0 0 0 8 8 8 8 0 4 3 3 3 13 13 13 13 3	315.000 258.500

B - Craters

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

General Warren Pierce has a bit of a problem. He's in charge of a new type of drone-delivered explosive and they've been testing it out in the Nevada desert, far enough from any population center to avoid civilian casualties and prying eyes. Unfortunately word has gotten out about these experiments and now there's the possibility of careless on-lookers, nefarious spies, or even worse – nosy reporters! To keep them away from the testing area, Warren wants to erect a single fence surrounding all of the circular craters produced by the explosions. However, due to various funding cuts (to support tax cuts for the you-know-who) he can't just put up miles and miles of fencing like in the good old days. He figures that if he can keep people at least 10 yards away from any crater he'll be okay, but he's unsure of how much fencing to request. Given the locations and sizes of the craters, can you help the General determine the minimum amount of fencing he needs? An example with three craters (specified in Sample Input 1) is shown below.

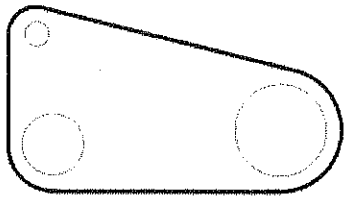

Figure 2: Three craters with a fence around them.

Input

The first line of input contains a single positive integer n ($n \leq 200$), the number of craters. After this are n lines specifying the location and radius of each crater. Each of these lines contains 3 integers x y r, where x and y specify the location of a crater ($|x, y| \leq 10000$) and r is its radius ($0 < r \leq 5000$). All units are in yards.

The input must be read from standard input.

Output

Display the minimum amount of fencing (in yards) needed to cordon off the craters. Output this number with exactly three decimal places, rounded.

The output must be written to standard output.

Sample Input	Sample Output
3 0 0 100 -60 200 40 350 50 150	1715.912

C - DRM Messages

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

DRM Encryption is a new kind of encryption. Given an encrypted string (which we'll call a DRM message), the decryption process involves three steps: Divide, Rotate and Merge. This process is described in the following example with the DRM message "EWPGAJRB":

Divide – First, divide the message in half to "EWPG" and "AJRB"".

Rotate – For each half, calculate its rotation value by summing up the values of each character (A = 0, B = 1, . . . Z = 25). The rotation value of "EWPG" is 4 + 22 + 15 + 6 = 47. Rotate each character in "EWPG" 47 positions forward (wrapping from Z to A when necessary) to obtain the new string "ZRKB". Following the same process on "AJRB" results in "BKSC".

Merge – The last step is to combine these new strings ("ZRKB" and "BKSC") by rotating each character in the first string by the value of the corresponding character in the second string. For the first position, rotating 'Z' by 'B' means moving it forward 1 character, which wraps it around to 'A'. Continuing this process for every character results in the final decrypted message, "ABCD".

Input

The input contains a single DRM message to be decrypted. All characters in the string are uppercase letters and the string's length is even and ≤ 15000.

The input must be read from standard input.

Output

Display the decrypted DRM message.

The output must be written to standard output.

Sample Input 1	Sample Output 1
EWPGAJRB	ABCD

Sample Input 2	Sample Output 2
UEQBJPJCBUDGBNKCAHXCVERXUCVK	ACMECNACONTEST

D - Game of Throwns

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

Daenerys frequently invents games to help teach her second grade Computer Science class about various aspects of the discipline. For this week's lesson she has the children form a circle and (carefully) throw around a petrified dragon egg.

The n children are numbered from 0 to $n-1$ (it is a Computer Science class after all) clockwise around the circle. Child 0 always starts with the egg. Daenerys will call out one of two things:

1. a number t, indicating that the egg is to be thrown to the child who is t positions clockwise from the current egg holder, wrapping around if necessary. If t is negative, then the throw is to the counterclockwise direction.
2. the phrase undo m, indicating that the last m throws should be undone. Note that undo commands never undo other undo commands; they just undo commands described in item 1 above.

For example, if there are 5 children, and the teacher calls out the four throw commands `8 -2 3 undo 2`, the throws will start from child 0 to child 3, then from child 3 to child 1, then from child 1 to child 4. After this, the `undo 2` instructions will result in the egg being thrown back from child 4 to child 1 and then from child 1 back to child 3. If Daenerys calls out 0 (or n, $-n$, $2n$, $-2n$, etc.) then the child with the egg simply throws it straight up in the air and (carefully) catches it again.

Daenerys would like a little program that determines where the egg should end up if her commands are executed correctly. Don't ask what happens to the children if this isn't the case.

Input

Input consists of two lines. The first line contains two positive integers n k ($1 \leq n \leq 30, 1 \leq k \leq 100$) indicating the number of students and how many throw commands Daenerys calls out, respectively. The following line contains the k throw commands. Each command is either an integer p ($-10000 \leq p \leq 10000$) indicating how many positions to throw the egg clockwise or undo m ($m \geq 1$) indicating that the last m throws should be undone. Daenerys never has the kids undo beyond the start of the game.

The input must be read from standard input.

Output

Display the number of the child with the egg at the end of the game.

The output must be written to standard output.

Sample Input 1	Sample Output 1
5 4 8 -2 3 undo 2	3

Sample Input 2	**Sample Output 2**
5 10 7 -3 undo 1 4 3 -9 5 undo 2 undo 1 6	2

E - Is-A? Has-A? Who Knowz-A?

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

Two familiar concepts in object oriented programming are the is-a and has-a relationships. Given two classes A and B, we say that A is-a B if A is a subclass of B; we say A has-a B if one of the fields of A is of type B. For example, we could imagine an object-oriented language (call it ICPC++) with code like that in Figure 3, where the class `Day` is-a `Time`, the class `Appointment` is both a `DateBook` and a `Reminder`, and class `Appointment` has-a `Day`.

```
class Day extends Time
{
   ...
}
```

```
class Appointment extends Datebook, Reminder
{
   private Day date;
   ...
}
```

Figure 3: Two ICPC++ classes.

These two relationships are transitive. For example if A is-a B and B is-a C then it follows that A is-a C. This holds as well if we change all the is-a's in the last sentence to has-a's. It also works with combinations of is-a's and has-a's: in the example above, `Appointment` has-a `Time`, since it has-a `Day` and `Day` is-a `Time`. Similarly, if class `DateBook` has-a `Year` then `Appointment` has-a `Year`, since `Appointment` is-a `DateBook`.

In this problem you will be given a set of is-a and has-a relationships and a set of queries of the form A is/has-a B. You must determine if each query is true or false.

Input

Input starts with two integers n and m, ($1 \leq n, m \leq 10000$), where n specifies the number of given is-a and has-a relationships and m specifies the number of queries. The next n lines each contain one given relationship in the form c_1 r c_2 where c_1 and c_2 are single-word class names, and r is either the string "is-a" or "has-a". Following this are m queries, one per line, using the same format. There will be at most 500 distinct class names in the $n + m$ lines, and all class names in the last m lines will appear at least once in the initial n lines. All is-a and has-a relationships between the given classes can be deduced from the n given relationships. Is-a relationships can not be circular (apart from the trivial identity "x is-a x").

The input must be read from standard input.

Output

For each query, display the query number (starting at one) and whether the query is true or false.

The output must be written to standard output.

Sample Input	Sample Output
5 5 Day is-a Time Appointment is-a Datebook Appointment is-a Reminder Appointment has-a Day Datebook has-a Year Day is-a Time Time is-a Day Appointment has-a Time Appointment has-a Year Day is-a Day	Query 1: true Query 2: false Query 3: true Query 4: true Query 5: true

F - Keeping On Track

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

Acmar and Ibmar are at war! You are in charge of a rail network that transports important supplies throughout the great state of Acmar during this delicate time. The rail system is made up of a set of rail lines which meet at various junction points. While there is no limit to the number of rail lines that can meet at a junction, the network is set up so that there is only one path between any two junctions. You've tried to argue for some redundancy in the system, i.e., extra rail lines so that there are two or more paths connecting some junctions, but it's wartime and budgets are tight.

However, this may soon change as you've just been given some terrible news from double agents working in Ibmar: within the next month enemy spies plan to blow up one of the junctions! Unfortunately, the exact junction is not known, but knowing your enemy well you are certain that they will undoubtedly strike the *critical junction*, specifically the junction whose removal disconnects the most pairs of other remaining junctions in the system. You don't have much time to act, so the most you can do is add one new line connecting two currently unconnected junctions, thereby reducing the number of disconnected pairs after the critical junction has been destroyed. Your job is to determine how to make the number of disconnected pairs as small as possible by adding in the best possible rail line.

Input

Input starts with a line containing an integer n ($2 \leq n \leq 10000$) indicating the number of rail lines in the system. Following that are n lines of the form i_1 i_2 indicating that a rail line connects junctions i_1 and i_2. Junctions are numbered consecutively starting at 0. All rail lines are two-way and no rail line appears more than once in the input. There is exactly one path between any two junction points given in the input.

The input must be read from standard input.

Output

Display two values n_1 and n_2, where n_1 is the number of pairs of junctions which will be disconnected when the enemy destroys the critical junction, and n_2 is the number of pairs of junctions still disconnected after you add in the best possible rail line. There will never be more than one critical junction.

The output must be written to standard output.

Sample Input 1	Sample Output 1
6 0 1 1 2 2 3 2 4 4 5 4 6	11 5

Sample Input 2	Sample Output 2
2 2 1 0 1	1 0

G - A Question of Ingestion

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

Stan Ford is a typical college graduate student, meaning that one of the most important things on his mind is where his next meal will be. Fortune has smiled on him as he's been invited to a multi-course barbecue put on by some of the corporate sponsors of his research team, where each course lasts exactly one hour. Stan is a bit of an analytical type and has determined that his eating pattern over a set of consecutive hours is always very consistent. In the first hour, he can eat up to m calories (where m depends on factors such as stress, bio-rhythms, position of the planets, etc.), but that amount goes down by a factor of two-thirds each consecutive hour afterwards (always truncating in cases of fractions of a calorie). However, if he stops eating for one hour, the next hour he can eat at the same rate as he did before he stopped. So, for example, if $m = 900$ and he ate for five consecutive hours, the most he could eat each of those hours would be 900, 600, 400, 266 and 177 calories, respectively. If, however, he didn't eat in the third hour, he could then eat 900, 600, 0, 600 and 400 calories in each of those hours. Furthermore, if Stan can refrain from eating for two hours, then the hour after that he's capable of eating m calories again. In the example above, if Stan didn't eat during the third and fourth hours, then he could consume 900, 600, 0, 0 and 900 calories.

Stan is waiting to hear what will be served each hour of the barbecue as he realizes that the menu will determine when and how often he should refrain from eating. For example, if the barbecue lasts 5 hours and the courses served each hour have calories 800, 700, 400, 300, 200 then the best strategy when $m = 900$ is to eat every hour for a total consumption of $800 + 600 + 400 + 266 + 177 = 2243$ calories. If however, the third course is reduced from 400 calories to 40 calories (some low-calorie celery dish), then the best strategy is to not eat during the third hour — this results in a total consumption of 1900 calories.

The prospect of all this upcoming food has got Stan so frazzled he can't think straight. Given the number of courses and the number of calories for each course, can you determine the maximum amount of calories Stan can eat?

Input

Input starts with a line containing two positive integers n m ($n \leq 100, m \leq 20000$) indicating the number of courses and the number of calories Stan can eat in the first hour, respectively. The next line contains n positive integers indicating the number of calories for each course.

The input must be read from standard input.

Output

Display the maximum number of calories Stan can consume.

The output must be written to standard output.

Sample Input 1	Sample Output 1
5 900 800 700 400 300 200	2243

Sample Input 2	Sample Output 2
5 900 800 700 40 300 200	1900

H - Sheba's Amoebas

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

After a successful Kickstarter campaign, Sheba Arriba has raised enough money for her mail-order biology supply company. "Sheba's Amoebas" can ship Petri dishes already populated with a colony of those tiny one-celled organisms. However, Sheba needs to be able to verify the number of amoebas her company sends out. For each dish she has a black-and-white image that has been pre-processed to show each amoeba as a simple closed loop of black pixels. (A loop is a minimal set of black pixels in which each pixel is adjacent to exactly two other pixels in the set; adjacent means sharing an edge or corner of a pixel.) All black pixels in the image belong to some loop.

Sheba would like you to write a program that counts the closed loops in a rectangular array of black and white pixels. No two closed loops in the image touch or overlap. One particularly nasty species of cannibalistic amoeba is known to surround and engulf its neighbors; consequently there may be amoebas within amoebas. For instance, each of the images in Figure 4 contains four amoebas.

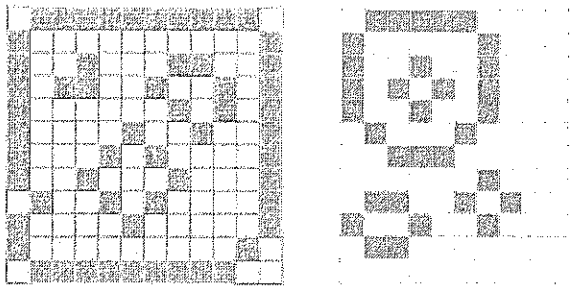

Figure 4: Two Petri dishes, each with four amoebas

Input

The first line of input contains two integers m and n, ($1 \leq m, n \leq 100$). This is followed by m lines, each containing n characters. A '#' denotes a black pixel, a '.' denotes a white pixel. For every black pixel, exactly two of its eight neighbors are also black.

The input must be read from standard input.

Output

Display a single integer representing the number of loops in the input.

The output must be written to standard output.

Sample Input 1	Sample Output 1
12 12 .##########. #..........# #..#...##..# #.##..#..#.# #......#.#.# #....#..#..# #...#.#....# #..#...#...# .#..#.#....# #....#.....# #.........#. .#########..	4

Sample Input 2	Sample Output 2
12 10 .#####.... #.....#... #..#..#... #.#.#.#... #..#..#... .#...#.... ..###..... ......#... .##..#.#.. #..#..#... .##.......	4

I - Twenty Four, Again

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

Yes, we know ... we've used Challenge 24 before for contest problems. In case you've never heard of Challenge 24 (or have a very short memory) the object of the game is to take 4 given numbers (the *base values*) and determine if there is a way to produce the value 24 from them using the four basic arithmetic operations (and parentheses if needed). For example, given the four base values 3 5 5 2, you can produce 24 in many ways. Two of them are: 5*5-3+2 and (3+5)*(5-2). Recall that multiplication and division have precedence over addition and subtraction, and that equal-precedence operators are evaluated left-to-right.

This is all very familiar to most of you, but what you probably don't know is that you can *grade* the quality of the expressions used to produce 24. In fact, we're sure you don't know this since we've just made it up. Here's how it works: A perfect grade for an expression is 0. Each use of parentheses adds one point to the grade. Furthermore, each inversion (that is, a swap of two adjacent values) of the original ordering of the four base values adds two points. The first expression above has a grade of 4, since two inversions are used to move the 3 to the third position. The second expression has a better grade of 2 since it uses no inversions but two sets of parentheses. As a further example, the initial set of four base values 3 6 2 3 could produce an expression of grade 3 – namely (3+6+3)*2 – but it also has a perfect grade 0 expression – namely 3*6+2*3. Needless to say, the lower the grade the "better" the expression.

Two additional rules we'll use: 1) you cannot use unary minus in any expression, so you can't take the base values 3 5 5 2 and produce the expression -3+5*5+2, and 2) division can only be used if the result is an integer, so you can't take the base values 2 3 4 9 and produce the expression 2/3*4*9.

Given a sequence of base values, determine the lowest graded expression resulting in the value 24. And by the way, the initial set of base values 3 5 5 2 has a grade 1 expression – can you find it?

Input

Input consists of a single line containing 4 base values. All base values are between 1 and 100, inclusive.

The input must be read from standard input.

Output

Display the lowest grade possible using the sequence of base values. If it is not possible to produce 24, display `impossible`.

The output must be written to standard output.

Sample Input 1	Sample Output 1
3 5 5 2	1

Sample Input 2	Sample Output 2
1 1 1 1	impossible

J - Workout for a Dumbbell

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

Jim Ratt has just joined a local fitness center. He's especially excited about a sequence of 10 machines that he cycles through three times for his workout. He has a fixed time which he spends on each machine, as well as a fixed recovery time after using a machine. Jim's not the brightest guy in the world, but in the absence of anything else even he would easily be able to calculate how long his workout would take.

But of course, Jim isn't the only person who uses the fitness center and wouldn't you know it but when Jim shows up there are always 10 other people there, each using one of the ten machines exclusively. Like Jim, each person has a fixed time they use on their machine as well as a fixed recovery time. This will sometimes cause Jim to have to wait for a particular machine, and Jim's usage sometimes results in the other people having to wait as well (though if both Jim and another person want to start using a machine at the same time, Jim is polite enough to let the other person go first). Jim has gone to the center often enough that he has a good idea what everyone's usage and recovery times are, but he has trouble determining how long it will take him to perform his workout. That's where you are going to flex your programming muscles.

Input

Input starts with a line containing twenty integers; the first two give Jim's usage and recovery time for machine 1, the next two give Jim's usage and recovery time for machine 2, etc. The next line contains 3 integers u r t; the first two values are the usage and recovery times for the person who is using machine 1, and t is the time when he/she first starts using the machine. The next 9 lines specify similar information for machines 2 through 10. All usage and recovery times are positive and ≤ 5000000 and all start times t satisfy $|t| \leq 5000000$. You should assume that Jim is ready to use machine 1 at time 0.

The input must be read from standard input.

Output

Display the time when Jim has finished his workout, i.e., the moment when he has finished his usage time on machine 10 for the third time (don't count the last recovery time for that machine).

The output must be written to standard output.

Sample Input	Sample Output
5 5 3 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 8 3 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0	100