Thirty-sixth Annual
University of Central Florida

High School Programming Tournament

Problems

Problem Name	**Filename**
Monkey Madness	monkey
Problematic Picasso	picasso
Team Naming	naming
A Very Hard Problem	hard
Catching Those Zs!	sleep
What are the Odds?	odds
Let's Groove	groove
Ovoo? What's That?	ovoo
Astronomy Rules!	astronomy
Dog Drama	dog
Campus Parade	parade
The Queen's Reign	queen
Arcane Artifacts	artifacts

Call your program file: *filename*.c, *filename*.cpp, *filename*.java, or *filename*.py

For example, if you are solving Monkey Madness:
Call your program file: monkey.c, monkey.cpp, monkey.java, or monkey.py
Call your Java class: monkey

Monkey Madness

Filename: `monkey`

The monkeys have set out for an invasion. They are planning to take over the world. After hijacking a fleet of gliders from Gnome Town (the gnomes were nonplussed because they were focused on a programming competition!), the mission went south and the flying monkeys went down. The monkeys have crash landed in the middle of a dense jungle, and are split up from each other.

Thankfully the walkie-talkies survived the crash, and one of the monkeys outside was able to climb a really tall tree to survey the area. The scout monkey reports the jungle layout as a rectangular grid, where each cell is either a crash survivor monkey or a plot of dense forestry. Each plot has a digit associated with it, representing the minimum strength of the machete required for the monkeys to chop through it. Monkeys are able to pass through adjacent plots that are chopped down and can chop through an adjacent plot if they have the necessary strength machete. Plots are adjacent if they share a complete side, not just a corner. A monkey can escape the jungle if they are able to move outside of the jungle.

The Problem:

The monkeys must get out of the jungle as soon as possible. Various machetes of different strengths are available to them. Each monkey will receive a machete of strength s. Can you determine the minimum s so that all monkeys can escape the jungle (so the remaining machetes can be reserved for gnome chopping)?

The Input:

The first line of input will contain a single positive integer, s, representing the number of scenarios. For each scenario, several lines follow. The first line of each scenario contains two integers, n and m ($3 \le n \le 1{,}000$; $3 \le m \le 1{,}000$), representing the number of rows and columns in the jungle, respectively. The following n lines each consist of a single string of m digits. A digit is 0 if it contains the position of a monkey, and 1-9 otherwise. It is guaranteed that the edges of the jungle contain no monkeys, and there is at least one monkey in the jungle.

The Output:

For each scenario, output a single line containing a single integer, the answer to the problem.

(Sample Input and Sample Output follow on next page)

Sample Input:

```
3
6 8
22211111
20211011
31114455
11110445
11110405
11119145
10 9
111111111
100111011
110011001
110111001
111111011
111111111
100110011
100101111
111101111
111111111
3 3
123
405
679
```

Sample Output:

```
4
1
2
```

Problematic Picasso

Filename: `picasso`

Picasso's daughter and protégé, Katie, has just finished painting several masterpieces and she is ready to sell them to the highest bidder. Well, there are only two bidders worthy of purchasing her art: Tess and Maxie. Tess has enough storage space in her nearby mansion to buy every painting Katie has made, but Maxie lives far away and has to ship the paintings on a luxury cargo plane. Maxie is very greedy, however, and she will be upset if she cannot completely fill her plane with Katie's paintings.

Since beauty is in the eye of the beholder, Tess and Maxie may offer different amounts of money for each painting. Katie, being the creator of these masterpieces, has the prerogative to choose who buys which painting. Wanting to avoid Maxie making a scene, she will sell Maxie exactly as many paintings as she can possibly store on her plane, and will sell the rest to Tess. Help Katie determine the maximum amount of money she can earn!

The Problem:

Given the amount of money Tess and Maxie will bid on each painting, as well as how many paintings Maxie will buy, determine the maximum amount of money Katie can earn if she sells her paintings optimally.

The Input:

The first line of the input will contain a single integer, *a*, representing the number of auctions. Each auction will be represented by multiple lines. The first line of each auction will contain two integers, *n* and *x* ($1 \le x \le n \le 1{,}000$), representing the number of paintings Katie has to sell, and the number of paintings Maxie will buy, respectively. The next line will contain *n* integers, $0 \le t_1, t_2, \ldots, t_n \le 10^5$, where t_i is the amount of money Tess will pay for the ith painting. The next line will contain *n* integers, $m_1, m_2, \ldots, m_n$ (each between 0 and 10^5, inclusive), where m_i is the amount of money Maxie will pay for the ith painting.

The Output:

For each auction, output on a new line the maximum amount of money Katie can make by selling her art such that Maxie gets exactly *x* paintings and Tess gets the rest.

(Sample Input and Sample Output follow on next page)

Sample Input:

```
2
5 3
1 2 3 4 5
3 1 9 2 3
4 2
2 0 2 2
1 2 3 4
```

Sample Output:

```
22
10
```

Team Naming

Filename: `naming`

Jacob, Daniel, and Lior are very excited to be on a team this year! Just one problem, though: They don't have a team name. They considered choosing a random proper noun out of the dictionary, but when they tried this, they landed on Hyperion, which nobody could pronounce. They were left with only one logical choice: the minimal length lowercase English string which contains each of their names as a subsequence and is lexicographically lower than all other valid strings of this length. A subsequence of string *s* is defined as a sequence that can be obtained from *s* by deleting some characters (possibly none or even all), without changing the order of the remaining ones. String *s* is lexicographically less than string *t* if *s* is a prefix of *t*, or if in the first position *i* the strings *s* and *t* differ, $s[i] < t[i]$.

Jacob, Daniel, and Lior ended up with the flawless team name "djaclnielobr". Their friends loved this idea, and wanted a team name generated with the same rules, but their names were much longer (Atharva, Anderson, and Ahmad). Soon, every team of three in the entire nation was emailing you asking for this hot new team name theme! Can you help them all?

The Problem:

Given three names, determine the shortest length string which contains all three names as a subsequence. If there are multiple of these, choose the lexicographically minimal (first alphabetical).

The Input:

The first line of the input will contain a single, positive integer, *t*, representing the number of teams asking for your help. Then, *t* lines will follow, each containing three space-separated strings of lowercase English letters (a-z). The length of any team member's name will be between 1 and 100 characters, inclusive.

The Output:

For each team, output the shortest length string which contains all three names as a subsequence. If there are multiple of the same length, output the lexicographically minimal one.

Sample Input:

```
3
daniel jacob lior
tyler kyle david
atharva anderson ahmad
```

Sample Output:

```
djaclnielobr
daktvidyler
anthmadersonva
```

A Very Hard Problem

Filename: `hard`

Jenna is the president of her school's Chemistry Bowl team, and one of her duties is to compile the list of practice problems her team will solve to train for the upcoming tournament. She wrote two problems, and wants to give her teammates the hardest one, but isn't sure which problem that is. Each problem has a length and complexity, and the difficulty of a problem is the product of these two values. Help Jenna determine the difficulty of the harder problem out of the two.

The Problem:

Given the length and complexity of both problems, determine the difficulty of the harder one.

The Input:

The first line of the input will contain a single, positive integer, d, representing the number of days Jenna runs chemistry practice. Then, d lines will follow, each with four integers, ℓ_1, c_1, ℓ_2, and c_2 ($0 \le \ell_1 \le 100$; $0 \le c_1 \le 100$; $0 \le \ell_2 \le 100$; $0 \le c_2 \le 100$), representing the length and complexity of the first problem, and the length and complexity of the second problem, respectively.

The Output:

For each practice day, output a line containing a single integer, the difficulty of the harder problem (or the difficulty of either problem if they tie with the same difficulty).

Sample Input:

```
3
4 9 5 6
2 6 3 4
2 0 2 2
```

Sample Output:

```
36
12
4
```

Catching Those Zs!

Filename: `sleep`

Dr. Doofenshmirtz has finished his latest invention: the Sleep-Inator!

The Sleep-Inator works by zapping a bystander. If that bystander's next sentence contains three or more 'z's (any mixture of uppercase or lowercase), then the entire tri-state area will fall asleep. Otherwise, the Sleep-Inator fails and nothing happens.

Perry the Platypus has arrived at Doofenshmirtz Evil Incorporated. Doofenshmirtz has just enough time to zap one person with his Sleep-Inator before Perry can press the self-destruct button. Perry does not know the person who will be zapped or what they will say. Your task is to tell Perry if the tri-state area will fall asleep given possible dialogue.

The Problem:

Given a possible bystander's sentence, tell Perry if that string contains three or more 'z's (any mixture of uppercase or lowercase).

The Input:

The first line of input will contain a single, positive integer, *t*, representing the number of bystanders to simulate. Each simulation will contain a string, *s*, to check on its own line. The string will be between 1 and 100 (inclusive) characters in length and only contain uppercase letters [A-Z], lowercase letters [a-z], spaces, and punctuation only in the set [! , . ?]. No string will contain leading or trailing spaces.

The Output:

Output a single line for each sentence. If there are three or more 'z's (any mixture of uppercase or lowercase) in the sentence, output "`Time to take a nap.`" Otherwise, output "`Perry saves the day!`"

Sample Input:

```
3
Zebras are my favorite animal!
The zombie zookeeper is surprisingly good with the zebras.
Fuzzy wuzzy was a bear.
```

Sample Output:

```
Perry saves the day!
Time to take a nap.
Time to take a nap.
```

What are the Odds?

Filename: `odds`

Malachi and his robot friend Robette are bored one afternoon and have a sudden and deep desire to analyze strings of characters. So, they invent a game nicknamed the "Unicode Character Fizzle". At the start of each game, Robette is given a string of two or more characters. Malachi is also given a string of one or more characters with length exactly one less than that of Robette's string.

Malachi challenges Robette to remove exactly one character from a position in her string. If after the removal, Robette's resulting string exactly matches Malachi's initial string, then Robette wins the game. If this does not occur, then Robette, like many of us, will have lost the game.

Robette finds this game trivial, until realizing that she is in fact a robot, and can only make completely random decisions. She is limited to removing a single character from a uniformly random position in the string (i.e. there is equal probability she removes the first character as the second character as the third character, and so on).

The Problem:

Please help Robette determine the chances that she successfully wins against Malachi.

The Input:

The first line of input will be a single, positive integer, *t*, representing the number of games played between Malachi and Robette. For each game, two lines of input will be given. The first will be the string, *r*, representing the string given to Robette. This string will consist of only lowercase letters and will have length between 2 and 10^5 inclusive. The second input line for each game will be the string, *m*, representing the string given to Malachi. This string will also consist of only lowercase letters and will have length exactly equal to one less the length of *r*.

The Output:

For each game, print the probability of Robette winning the game on one line as a reduced fraction in the form of `p/q`, where *p* and *q* are both integers and are separated by a single '`/`' character with no spaces. A reduced fraction is one in which the numerator and denominator share no common factors other than the number 1.

(Sample Input and Sample Output follow on next page)

Sample Input:

```
3
aba
ab
noooooonoo
nooooonoo
brother
please
```

Sample Output:

```
1/3
3/5
0/1
```

Let’s Groove

Filename: `groove`

DJ Saturday Nite loves writing music for improvised song and dance numbers in high schools across America. She isn't afraid to make her music wild and exciting, although it is simultaneously well-structured, with an exposition, development, and recapitulation.

A song may be represented by an array of positive integers, where each integer is a note. The *flow* of a song is defined as the largest positive integer, k, such that the subsequence [1, 2, 3, ..., k-1, k, k, k-1, ..., 3, 2, 1] exists in the array. For example, the array [1, 4, 2, 3, 1, 3, 2, 3 1] contains the subsequence [1, 2, 3, 3, 2, 1], so its flow is 3. If no such k exists, the flow of the song is 0. Note that a subsequence of an array can be obtained by deleting some elements from the array (possibly none), without changing the order of the remaining elements.

DJ Saturday Nite has written many songs, and she needs your help to determine the flow of each.

The Problem:

Given a song, determine its flow.

The Input:

The first line of input contains a single, positive integer, t, representing the number of songs. For each song, the first line of that song contains an integer, n ($1 \leq n \leq 10^5$), representing the number of notes in the song. The following line contains n integers, a_i ($1 \leq a_i \leq 10^5$), representing the notes of the song, respectively.

The Output:

For each song, output a single integer on its own line: the flow of the song.

Sample Input:

```
3
9
1 4 2 3 1 3 2 3 1
5
1 3 4 2 1
6
5 4 3 2 1 2
```

Sample Output:

```
3
1
0
```

Ovoo? What's That?

Filename: `ovoo`

An Ovoo (oh-voo) is a sacred stone heap that serves as an altar or shrine in Mongolian folk religion. (Fictional information starts here). Each Mongolian clan constructs their own Ovoos. Every day is either a work day or a worship day. During a workday, a new Ovoo is built; during a worship day, the favor of the gods towards a clan (initially zero) increases by the *total* amount of Ovoos constructed up to that day. The favor of each clan only depends on the amount of Ovoos they constructed, and no Ovoos of other clans are considered.

The Problem:

Given a clan's schedule, determine the favor of the gods on each worship day.

The Input:

The first line of input will contain a single, positive integer, *c*, representing the number of clans. For each clan, several input lines follow. The first line of input contains a single integer, *n* ($2 \le n \le 100$), representing the number of days. Then, *n* lines follow, each containing a single string: `WORK` or `WORSHIP`, depending on whether the day is a work day or a worship day. It is guaranteed that the input for each clan begins with a workday and ends with a worship day.

The Output:

For each clan, output a single line containing "`Clan #`*`i`*`:`" where *i* is the number of the clan in the input (starting at 1). Then for each worship day, output the favor of the gods on its own line. Leave a blank line after the output for each clan.

(Sample Input and Sample Output follow on next page)

Sample Input:

```
2
10
WORK
WORSHIP
WORK
WORSHIP
WORK
WORSHIP
WORK
WORSHIP
WORK
WORSHIP
4
WORK
WORK
WORSHIP
WORSHIP
```

Sample Output:

```
Clan #1:
1
3
6
10
15

Clan #2:
2
4
```

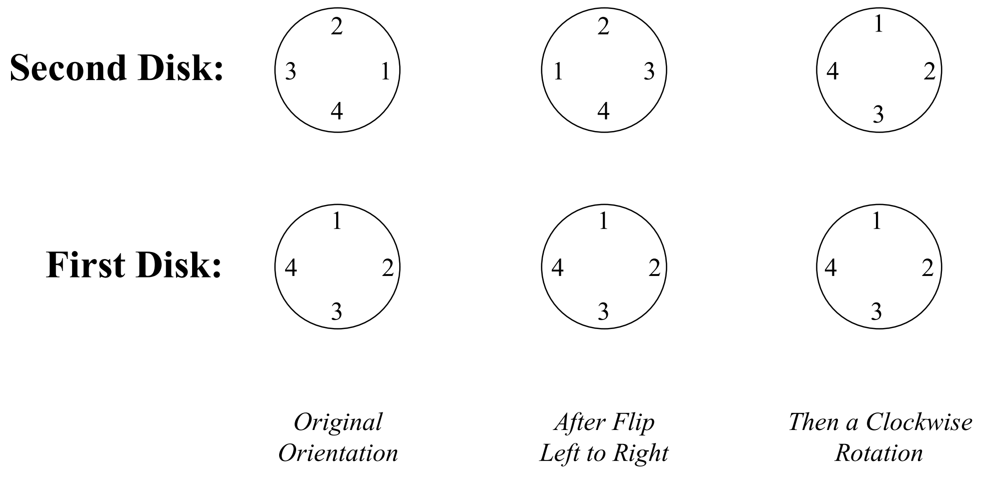
Astronomy Rules!

Filename: `astronomy`

Geoff, a student attending the University of Crispy Fries, recently aced his astronomy class. Obviously, he is now a well-renowned expert in the field. During his studies, he found that if the energy from the large star Betelgeuse is focused in just the right way, Geoff can cause an explosion on the surface of the earth large enough to eliminate all greenhouse gases in the atmosphere, therefore saving the planet and humanity as a whole.

Luckily, he has just the tools to pull off this amazing feat. In his basement sit two circular disk lenses atop each other. Each disk has the numbers 1 through *n* written around the outside in some jumbled order. The disks must be placed flat on the basement, one on top of the other. The first disk is extremely heavy and Geoff is unable to lift it off of the ground. The second disk is light enough for Geoff to rotate, and even flip, to his liking. Geoff's goal is to align the disks so that the numbers on the sides of the disks line up perfectly with each other (i.e. the 1 of the first disk is below the 1 of the second disk, the 2 of the first disk is below the 2 of the second disk, and so on).

For example, given the following original orientation of two circular disks with 4 numbers written on them, the second disk could first be flipped left to right and then given a clockwise rotation. This results in the second disk matching the first.



If Geoff is able to line up these disks, he will then bore a vertical hole through his entire home from the roof all the way through the ceiling of the basement. At the time of the winter solstice, the light from Betelgeuse will strike these disks, causing an explosion that will wipe away all pesky greenhouse gases lingering in the atmosphere in an event called the "Harmless Star Photon Transform."

The Problem:

Help Geoff determine if he can save the planet, or if humanity is truly doomed.

The Input:

The first line of input will contain a single, positive integer, *t*, representing the number of parallel universes that Geoff will help. Each universe will be represented by multiple lines. For each universe, it will begin with a line consisting of an integer, *n* ($1 \le n \le 10^5$), representing the number of numbers written on each of the disks. Then the next line will be a sequence of *n* integers, each distinct and between 1 and *n* (inclusive), specifying the numbers on the outside of the first disk in clockwise order. The next line will have the same format as the previous and will specify the numbers on the outside of the second disk in clockwise order. Note: a flip of the second disk can be thought of as a reversal of the second sequence, and a rotation can be thought of as a circular shift of the second sequence.

The Output:

For each universe, if Geoff can save the Earth, print "`Saved`" on one line. If not, print "`Doomed`" on the one line instead.

Sample Input:

```
4
4
1 2 3 4
2 1 4 3
3
1 2 3
1 3 2
5
2 1 3 5 4
3 2 1 4 5
7
1 5 3 2 4 7 6
7 6 1 5 3 2 4
```

Sample Output:

```
Saved
Saved
Doomed
Saved
```

Dog Drama

Filename: `dog`

You were walking your dog when suddenly you got a very important phone call! You stop in the middle of a park to take it, while your dog runs around you as you hold her leash.

As your very important phone call finishes, you realize you made a terrible mistake! You forgot your very special dog loves flowers. She loves flowers so much that she will always eat them if she can reach them. She can reach any flower as long as the distance to the flower does not exceed the length of her leash.

This is very sad since flowers contribute greatly to the beauty of the park. You want to know how many flowers are within reach of your dog when you stopped to take the call.

The Problem:

Given the length of your dog's leash, the initial locations of the flowers, and where you were standing during the phone call, determine how many flowers were within your dog's reach.

The Input:

The first line of input will contain a single, positive integer, t, representing the number of parks to test. The first line of each park will contain two space-separated integers, n ($1 \leq n \leq 10^4$) and ℓ ($1 \leq \ell \leq 10^4$), where n is the number of flowers in the park and ℓ is the length of your dog's leash.

The next n+1 lines will contain a coordinate point represented with two space-separated integers, x_i and y_i, each between -10^4 and 10^4 inclusive. The first of these lines is the position that you were standing, and the remaining n lines each represent the position of a flower. No two points are equivalent.

The Output:

For each park, output a single line containing the number of flowers your dog could have reached.

(Sample Input and Sample Output follow on next page)

Sample Input:

```
2
4 10
0 0
1 1
0 10
-1 5
100 -10
2 50
50 50
51 53
0 0
```

Sample Output:

```
3
1
```

Campus Parade

Filename: `parade`

An annual parade is taking place on the UCF campus. Each year, a line of *n* people, numbered from 1 to *n* from front to back, marches through campus. Initially, the line begins in increasing order from front to back [1, 2, …, *n*]. Along the way, they will travel through *m* doors. Every time they encounter a door, the first person in the line holds the door open for everyone else. Then, everyone in the line after the first passes through the door in order. Finally, after everyone is through, the person holding the door rejoins the line in the back. However, if the first person is all alone in the parade, then that parader simply passes through the door by their lonesome.

The event organizer wants to know who the first person in the line will be at the end of the parade. The university has chosen you to write a program that completes this task.

The Problem:

Given the number of people and number of doors, determine the first person in line at the end of the parade.

The Input:

The first line of input will be a single, positive integer, *y*, representing the number of years the parade will run for. For each year, there will be one line of input with two integers, *n* and *m* ($1 \le n \le 1{,}000$; $1 \le m \le 1{,}000$), representing the number of people in the line and the number of doors they will go through, respectively.

The Output:

Output *y* lines with the i^{th} line containing a single integer, the number of the person in the front of the line at the end of i^{th} parade.

Sample Input:

```
3
5 2
3 1
4 100
```

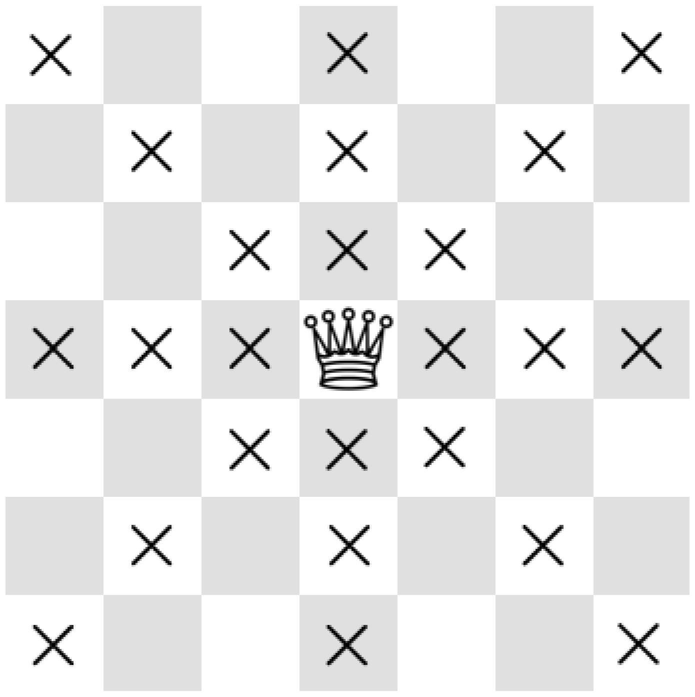
Sample Output:

```
3
2
1
```

The Queen's Reign

Filename: `queen`

Beth has recently started playing chess, and unsurprisingly, has ever since been fascinated by the chess piece known as the queen. Who wouldn't be? The piece can travel in all 8 directions (vertically, horizontally, and diagonally), as far as it wants (given there are no obstacles and the destination is within the bounds of the board), and can capture any opposing piece blocking its path. What a powerful piece!



The queen's reign can be defined as the set of squares that the queen can move to in a single turn on an otherwise empty board. Beth wants to know everything there is to know about the queen, starting with exactly how many squares that are not under the queen's reign on a given chessboard. In other words, she wants to know the number of squares where another piece can be placed without being captured by the queen in a single move. The chessboard Beth is considering are $(2k + 1)$ by $(2k + 1)$ square boards, where k is a non-negative integer, and the queen is at the center of each board. For example, the image above depicts a board where $k = 3$, and there are 24 squares not under the queen's reign (squares not marked by 'X' or the queen).

The Problem:

Given a non-negative integer, k, which defines a chessboard as described above, output the number of squares not under the queen's reign.

The Input:

The first line of the input will contain a single, positive integer, *n*, representing the number of different chessboards Beth is curious about. The next *n* lines will each contain an integer, *k*, as described above ($0 \leq k \leq 10^4$), representing each of these chessboards.

The Output:

For each of the chessboards, output on its own line a single integer, representing the number of squares not under the queen's reign.

Sample Input:

```
2
3
1
```

Sample Output:

```
24
0
```

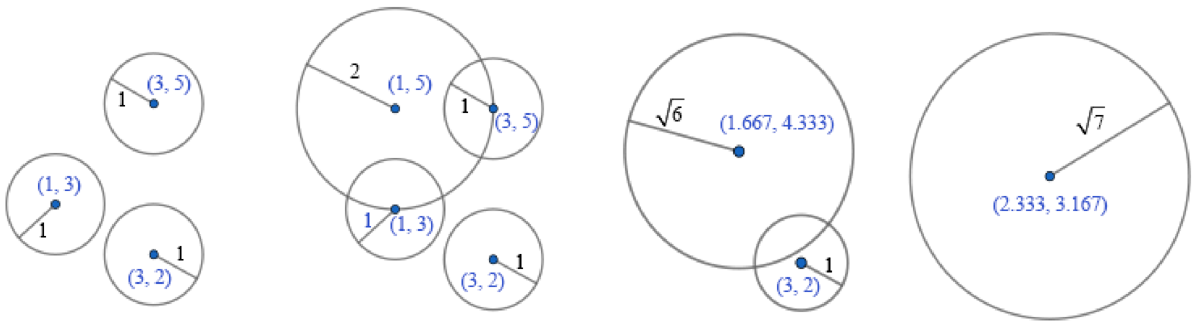
Arcane Artifacts

Filename: `artifacts`

Rumor has it that there is a set of arcane artifacts which hold unimaginable power. Marcus, a prestigious treasure hunter, is determined to collect them. These artifacts, however, are quite unstable and will relocate periodically. Through rigorous calculation, Marcus has been able to locate the coordinates of where the next several artifacts will appear and wants to collect as many as possible. To conserve time and resources, Marcus wants to construct a team to gather these artifacts such that each person pursues exactly one artifact.

Artifacts will generate at new locations and no two artifacts will ever generate at the same time as per the Hyper-Sorcery Particle Theorem. When a new artifact appears, it has a magical field that can be represented as a circle. When this field interacts with other artifacts' fields, the respective artifacts will begin to attract one another and merge into a single, more powerful artifact. The new position of these artifacts is the simple average position and the area of the new field is equal to the sum of the areas of each field. Two artifacts interact if their areas of magical fields overlap or are within 10^{-5} of each other.

For example, assume one artifact appears at (1, 3), another at (3, 5), followed by a third at (3, 2). All three of these artifacts' fields have a radius of 1. Since none of these artifacts have interacting fields, none of them merge. A fourth artifact then appears at (1, 5) with a field radius of 2. This fourth artifact now interacts with the first and the second, resulting in them merging into one artifact. Taking the average position of these three results in approximately (1.667, 4.333), which is the location of the new artifact. The field area of this new artifact will then be the sum of the three separate artifacts: $\pi(1)^2 + \pi(1)^2 + \pi(2)^2 = 6\pi$. So the radius of the new artifact will be $\sqrt{6}$. This new artifact is now large enough to interact with the third artifact, resulting in a single artifact with a position of (2.333, 3.167) and a radius of $\sqrt{7}$. This scenario is modeled below.



After the first 3 artifacts appear | *After the fourth artifact appears* | *After the first merge* | *After the second merge*

The Problem:

Given a list of artifacts' coordinates and their radii of influence, determine the minimum number of people needed in order to collect all artifacts after every artifact has been generated and merged.

The Input:

The first line of input will contain a single, positive integer, *t*, representing the number of expeditions. Each expedition will begin with a line containing a single integer, *n* ($1 \leq n \leq 100$), indicating the number of artifacts. The next *n* lines will each contain three integers, x_i, y_i, and r_i ($-1{,}000 \leq x_i \leq 1{,}000$; $-1{,}000 \leq y_i \leq 1{,}000$; $1 \leq r_i \leq 1{,}000$), where x_i is the *x*-coordinate of the *i*th artifact, y_i is the *y*-coordinate of the *i*th artifact and r_i is the radius of influence of the *i*th artifact. Artifacts appear in the given order, one by one. The next artifact appears only after the existing artifacts stop interacting. It is guaranteed that for any point in time, no artifacts share the same positions.

The Output:

For each expedition, output one number: the number of people needed to complete Marcus's expedition.

Sample Input:

```
3
4
1 3 1
3 5 1
3 2 1
1 5 2
4
1 3 1
0 0 1
2 0 1
5 8 4
5
0 0 4
1 0 1
-3 4 2
1 -5 5
-1 -3 6
```

Sample Output:

```
1
3
1
```