UCF Programming Team Practice
September 29, 2018
Developmental Team
Individual Problem Set

Problem Name	Filename
The Little Bird	bird
Simi Circles	circles
Number of Factors	factors
Family Tree	family
Enraged Fowl	fowl
Happy Sets	happy
Get out of this Maze!	maze
Come Minion!	minion
Degrees of Separation	relatives
Arup's Steps	steps

The Little Bird

Filename: `bird`

One day, while playing in her backyard, a little girl came across an injured baby bird. Like any good kid would do, she picked it up and took it home and decided to take care of it until it got better. After it healed, she decided to give it flying lessons in her backyard. The girl's back yard is well-fenced, and the bird is safe inside the fence. However, she lives near the woods, which are filled with cats and foxes and thorny trees and many other things that might hurt the helpless little bird, and if the little bird were to land on or outside the fence, there's no telling what might happen to it! As a result of these dangers, the girl knows that until it is a strong flier, it will have to remain confined to the yard, so she wants to make sure that the yard is absolutely safe for the bird to practice in.

She knows that because the bird is still learning, it can fly at most a certain distance away from her before getting tired and landing. However, the bird may fly in any direction away from her when she releases it, and the yard is considered unsafe if flying in any direction could result in the bird landing on or outside the fence. The girl will be standing inside of her yard, at coordinates (0, 0) when she releases the bird. She also knows that her parents love geometry and have fenced their yard in such a way that it is a polygon whose internal angles are all less than 180 degrees.

The Problem:

Given the coordinates of the corners of the fence in clockwise order, and the maximum distance that the little bird can fly, determine whether or not a given yard is safe for the bird.

The Input:

The first line will contain a single, positive integer, t, representing the number of yards to be checked. Each yard will begin with a line containing two integers, r and n ($1 \leq r \leq 1{,}000$; $3 \leq n \leq 100$), representing the maximum distance that the little bird can fly and the number of corners in the fence, respectively. The next n lines will each contain two integers, x and y ($-1{,}000 \leq x \leq 1{,}000$; $1{,}000 \leq y \leq 1{,}000$), representing the coordinates of a corner of the fence.

The Output:

For each yard, output "`Yard #`*`i`*`: `*`m`*" where i is the number of the yard (starting with 1) and m is the message "`Fly away!`" if it is safe for the bird to fly in the given fencing layout or the message "`Better not risk it.`" if not.

Sample Input:

```
2
10 4
8 8
8 -8
-8 -8
-8 8
3 4
-6 7
7 8
9 -6
-7 -8
```

Sample Output:

```
Yard #1: Better not risk it.
Yard #2: Fly away!
```

UCF Local Contest — September 1, 2012

Simi Circles

filename: `circles`

Arup has a step-daughter, Simi, at home, so he's frequently in charge when she's swimming with her friends. Unfortunately, after she's done, Simi will run inside the house without completely drying herself. As she scurries through the house to the bathroom, she leaves drops of water that fall on the floor. Naturally, Arup's wife, Anita, blames him for not properly watching the kids and making sure they properly dry themselves before entering the house. Anita will say that the wood was ruined by the water damage. But, of course, Arup realizes that not all of the wood is ruined. Thus, he only wants to take responsibility for the area of the wood floor that has water on it.

When Simi runs through the house, each drop of water hits the ground and forms a perfect circle. Furthermore, she runs fast enough that any drop at most intersects with the previous drop and no others. Sometimes a drop does not intersect with the previous drop. This means that any individual drop can at most intersect with two drops: the one before it and the one after it. Assume that each drop (circle) covers some area of the wood floor. In particular, assume that a circle will not be completely encompassed (covered) by another circle.

The Problem:

Given a list of circles, where a circle in the list may only intersect the previous circle on the list and the subsequent circle on the list, determine the total area that the circles cover.

The Input:

The first input line contains a single positive integer, n $(1 \le n \le 100)$, indicating the number of Simi circle scenarios to consider. Each of the n input sets follows. The first line of each input set contains only an integer, c $(1 \le c \le 100)$, representing the number of drops for the input set. The following c lines contain information about each drop, one drop per line, in the order that the drops occurred. Each of these lines contains exactly three real numbers (each separated from the next by a single space): x $(-3000 < x < 3000)$, y $(-3000 < y < 3000)$, and r $(0 < r < 10)$, representing (respectively) the x and y coordinates of the center of the drop (in cm) and the radius of the drop (in cm).

The Output:

For each Simi circle scenario, first output "`Set #i: `" where `i` is the input set number, starting with 1. Follow this with a single positive real number rounded to 2 decimal places, representing the total area (in cm^2) covered by all of the drops for the Simi circle scenario. Thus, if the actual area is 31.674 cm^2, output 31.67 and if the actual area is 31.675 cm^2, output 31.68.

Leave a blank line after the output for each test case. Follow the format illustrated in Sample Output.

(Sample Input/Output on the next page)

Sample Input:

```
3
2
0 0 1
0 2 1
2
0 0 5
6 0 5
1
0.1 0.1 0.1
```

Sample Output:

```
Set #1: 6.28

Set #2: 134.71

Set #3: 0.03
```

Note: If you need to use pi in your program, use the value 3.1415926535898.

Number of Factors

Filename: *factors*

Arup has a company that sells soda and he also happens to like numbers that have many factors. For example, the number 6 has 4 factors: 1, 2, 3 and 6. Thus, he can package a group of 6 sodas in 4 rectangular arrangements: 1 x 6, 2 x 3, 3 x 2 and 6 x 1. (He considers all four of these distinct because the vertical and horizontal arrangements look different.)

Arup has realized that consumers will buy more soda whenever it's repackaged in a different shape. So he has plans to sell packages of soda of a particular size (number of sodas) and has set dates he'll keep the same size but change the packaging arrangement. Naturally, people are less likely to buy very large packages of soda, so Arup's preference is to minimize the number of sodas in a package, given the number of arrangements possible. Thus, if Arup wants 4 possible arrangement of packages of a fixed size, he prefers a package of 6 sodas over 8 sodas (8 sodas can be arranged in 4 ways also: 1 x 8, 2 x 4, 4 x 2 and 8 x 1).

At various times, Arup will ask his staff to find the smallest integer with a particular number of factors. One other restriction Arup has is that he doesn't like numbers with more than two distinct prime factors, so his actual request for you is to find the smallest integer with no more than two distinct prime factors that has a particular number of factors.

The Problem

Given a positive integer n, greater than 1, determine the smallest integer with two or fewer distinct prime factors that has exactly n distinct positive factors.

The Input

The first line of input will contain a single positive integer, n ($n \leq 300$), the total number of input cases. Each of the following n lines contains a single positive integer, d ($2 \leq d$), representing the input for the test case. Each of these inputs will be such that there exists an integer less than 10^{15} with no more than two distinct prime factors that has exactly d distinct factors.

The Output

For each input case, d, output the smallest integer with two or fewer distinct prime factors that has exactly exactly d distinct positive factors on a line by itself.

Sample Input

```
3
2
4
5
```

Sample Output

```
2
6
16
```

Family Tree

Filename: `family`

You've been studying a lot of family trees lately, and have started to get confused as to who's in whose family. The connections span many generations, and a lot of people are involved. Since you are about to go to a family reunion, and want to at least appear as if you know who's in your own family, you decide to write a program to help you sort it all out.

The Problem:

Given a description of a family tree, your task is to determine whether or not two people are related. For this problem, two people are considered related if there is a path between the two in the family tree. A "path" is any series of parent-child (or child-parent) relationships that connect people in the tree (not only blood relatives are related). It is guaranteed that each child will have exactly two distinct parents, and each family tree will be valid, meaning that no child will be his/her own ancestor (luckily for you, there is no time travel allowed for this problem!).

The Input:

There will be multiple family trees. Each family tree will begin with an integer, n ($1 \leq n \leq 100$), on a line by itself, indicating the number of connections to be listed. This will be followed by exactly n lines, each of the form *parent1 parent2 child*, indicating that *parent1* and *parent2* are the parents of *child*. Each name will be separated by a single space. Each name will consist of only upper and lowercase letters, and no name will exceed 80 letters. In addition, each different person will have a unique, case-sensitive name. Each relationship will be unique (there will be no repeated relationships within a single family tree). This will be followed by a line of the form *name1 name2*, indicating the two people whose relation you are to determine. Both names will follow the same conventions as before, and will be separated by a single space. The final family will be indicated by $n = 0$, and should not be processed.

The Output:

For each family tree, you should output a single line beginning with "`Family #x:`" where *x* is the family tree being processed (beginning with 1). This should be followed by one space, then either "`Related.`" or "`Not Related.`" to indicate whether or not the two people are related.

(Sample Input and Sample Output are on the following page)

Sample Input:

```
2
Barbara Bill Ted
Nancy Ted John
John Barbara
3
Lois Frank Jack
Florence Bill Fred
Annie Fred James
James Jack
1
John Susan Billy
John Susan
2
Karen Roger Christopher
Karen Roger Michael
Christopher Michael
0
```

Sample Output:

```
Family #1: Related.
Family #2: Not related.
Family #3: Related.
Family #4: Related.
```

Enraged Fowl

Filename: `fowl`

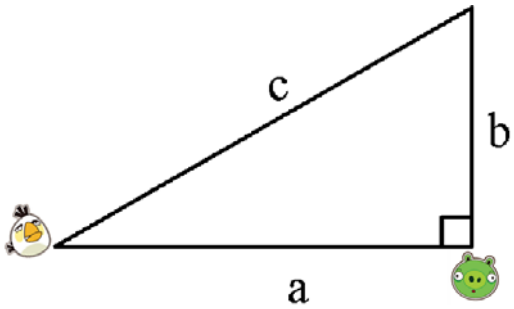
Brandon is playing the latest game sensation, Enraged Fowl! His favorite fowl to use in this game is the one that "flies" in a straight line and then drops a bomb straight down on a target. Unfortunately, he is not very good at dropping the bomb so he wants to practice outside of the game. He wants you to help him! For his practice, he will consider only games where the target is in the same place (at the same height) as himself. He will provide you the distance that the target is from his position, the straight-line distance that he will fling his fowl through the air and the distance the bomb will drop. You need to determine whether that combination will work and hit the target!

The Problem:

Given three values representing the target distance, the flying distance, and the bomb drop length, determine whether Brandon can hit the target.

The Input:

The input will begin with a line containing a positive integer, t ($1 \leq t \leq 100$), representing the number of scenarios to check. Each scenario consists of a single line containing three positive integers, representing the distance the fowl flew, the distance of the target from the starting point of the fowl, and the distance the bomb dropped (straight down). Note that the three distances will be given in any order and each will be separated by a single space. All of the distances are between 1 and 100, inclusive.



The Output:

For each scenario, output a line "`Target #x: m`" where x is the number of the target in the input (starting from 1) and m is "`YES`" if the Brandon can hit the target or "`NO`" otherwise.

Sample Input:

```
5
1 35 68
59 79 25
36 15 39
28 82 46
43 96 92
```

Sample Output:

```
Target #1: NO
Target #2: NO
Target #3: YES
Target #4: NO
Target #5: NO
```

Happy Sets

Filename: `happy`

You have recently been awarded the prestigious title of Uncontested Calculus Führer of Orlando, FL. In an effort to prove to your colleagues that you deserve this title, you have invented an entirely new set of numbers, which, given your generally genial mood, you have aptly titled a 'Happy Set.' A happy set is any set of numbers A that can be rearranged in some way to meet the following property:

$$A_0 + A_1 - A_2 * A_3 \,/\, A_4 + A_5 - A_6 * A_7 \,/\, A_8 + A_9 - \cdots = An\ Integer\ Value$$

taking note that normal order of operations is *not* followed here, but each calculation is carried out sequentially. That is, first add A_o to A_1, then subtract A_2 from that value, then multiply A_3 by that value, and so on. Also note that if at any point along the way a calculation results in a value that is not an integer (or tries to make an illegal operation), then the set is a Sad Set.

It is guaranteed that the result x of each calculation will fall in the range $-2147483648 \leq x \leq 2147483647$.

The Problem

Although you are generally a happy person, you are also very lazy. Given this, you want to write a program that will solve this problem for you. Given a set of numbers A, you must determine whether any permutation of that set A results in a Happy Set.

The Input

The input file to this program will begin with an integer, $n \geq 1$, indicating the number of data sets to be processed.
On each of the next n lines one test case will appear. The first integer in each list, k, $2 \leq k \leq 9$, will indicate the number of values for that test case, and will be followed by k integers. All k integers are guaranteed to be positive, unique, and will fit in an integer data type.

The Output

Each data set should output one of the following lines:

```
Set i is a Happy Set =)
Set i is a Sad Set =(
```

where i is the number of the data set (starting at 1).

Sample Input

```
2
6 3 4 1 5 2 12
6 223 251 41 434 201 277
```

Sample Output

```
Set 1 is a Happy Set =)
Set 2 is a Sad Set =(
```

Get Out of This Maze!!!

Filename: `maze`

Getting out of mazes can be quite tricky! So tricky in fact, that you've decided that it's better to write a program to determine the fastest way out of a maze than to try to wing it. For the purposes of this problem, a maze will be described as a two dimensional grid, where each square is one of the following types:

Outside Border Square (`~`)
Illegal Square to Travel on (`X`)
Valid Square to Travel on, not on the Outside Border (`-`)
Your Initial Starting Square (`S`)

Here is an example of a maze with 5 rows and 7 columns:

```
~~~~~~~
~XXXXX~
~X-S-X~
~--X-X~
~~~~~~~
```

We consider reaching any Outside Border Square as getting out of the maze. In a single move, you may move up, down, left or right. Thus, for this maze, one of the paths that is fastest to get out would be to move right, move down and move down again, for a total of three moves. (One could also move left, down and down and get out in three moves as well.)

The Problem

Given a maze as described previously, determine whether or not there is a way to escape to the boundary, and if so, what the shortest distance to escape the maze to any of the boundary positions is. At each move, you may move up, down, left or right from your previous spot, so long as the new spot isn't forbidden.

The Input

The first line of the input file will contain a single positive integer, c $(c \leq 100)$, representing the number of input cases. The input cases follow, one per line. The first line of each input case will contain two positive integers, $r(3 \leq r \leq 300)$, and $c(3 \leq c \leq 300)$, representing the number of rows and columns, respectively, in the maze. The following r lines will contain strings of exactly c characters, describing the contents of that particular row using the symbols described above (`~`, `X`, `-`, `S`). You are guaranteed that the first and last rows and first and last columns will only contain the border character, `~`. You are guaranteed that this character will not appear anywhere else in the grid. Exactly one non border character will be an `S`, so there will always be exactly one starting location. Finally, the rest of the squares will either be `-`, to indicate that that square is valid to travel to, or `X`, to indicate that you may not travel to that square.

The Output

For each case, if it's possible to reach the border of the maze, output the fewest number of steps necessary to reach any border square. If it's not possible, output -1.

Sample Input

```
3
3 3
~~~
~S~
~~~
5 6
~~~~~~
~XXXX~
~XS-X~
~-XX-~
~~~~~~
5 7
~~~~~~~
~XXXXX~
~X-S-X~
~--X-X~
~~~~~~~
```

Sample Output

```
1
-1
3
```

UCF Local Contest — August 31, 2013

Come Minion!

filename: `minion`

Welcome treasure hunter! I am the great and powerful interplanetary ninja and you are my minion. We must hurry to save this planet from the evil, "Dialog Not Found". We must travel across the land so that I may prevent "Dialog Not Found" from their evil plans. Also just so you know, minion, I am unable to triumph over a great many trials. Some of these include being looked at, handshakes, and even stairs.

The Problem:

Given a map of locations, routes between locations, and the trials which exist along routes, help the interplanetary ninja reach their target. You must avoid any of the trials that the ninja is unable to triumph over. Then tell the interplanetary ninja if they are able to reach their target and save the world.

The Input:

The first input line contains a positive integer, *m*, indicating the number of maps to check. Each map will start with an integer on a new line, t ($0 \leq t \leq 50$), that describes the number of trials that the ninja is unable to accomplish. On the next t lines these trials are listed, one per line. The following line will contain two integers, n ($2 \leq n \leq 30$) and e ($0 \leq e \leq 500$). n indicates the number of locations on the map (numbered 0 through n-1) and e indicates the total number of routes between locations on the map. Assume the ninja's starting location number is 0 and the ninja's target location is n-1.

The following e lines each describe a route between a pair of locations, and the trial on that route. These lines will consist of two integers L_a and L_b and a string Q. The ninja can travel between location L_a and location L_b, or between L_b and L_a, as long as he does not have the trial (Q) on that route. For example, if ninja has the trial "xyz" and the route also has the trial "xyz", then ninja cannot travel on that route. Assume that there is at most one route between any two locations and exactly one trial for a route.

All locations are numbered from 0 to n-1, inclusive. All trials are named using only lowercase letters, 1 to 20 in length. If a trial is not on the "unable to accomplish" list of ninja, then the ninja will be able to accomplish it. Successive values on a line are separated by exactly one space. There are no leading or trailing spaces on any line.

The Output:

For each map, output a line that contains only a 1 if the ninja can reach his target and a 0 if the ninja is unable to reach his target. Each answer must be on a separate line.

(Sample Input/Output on the next page)

Sample Input:

```
2
3
stairs
talking
staring
4 5
0 3 talking
0 1 abc
0 2 xyz
1 3 stairs
2 3 staring
3
fire
water
people
4 5
0 1 abc
0 2 water
1 2 fire
1 3 xyz
2 3 abc
```

Sample Output:

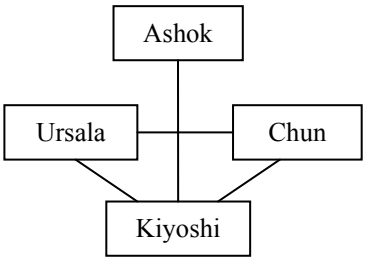
```
0
1
```

Problem I
Degrees of Separation
Input File: relatives.in

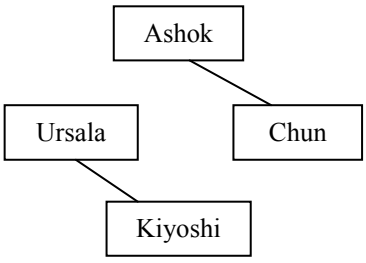
In our increasingly interconnected world, it has been speculated that everyone on Earth is related to everyone else by no more than six degrees of separation. In this problem, you must write a program to find the maximum degree of separation for a network of people.

For any two people, the degree of separation is the minimum number of relationships that must be traversed to connect the two people. For a network, the maximum degree of separation is the largest degree of separation between any two people in the network. If there is a pair of people in the network who are not connected by a chain of relationships, the network is disconnected.

As shown below, a network can be described as a set of symmetric relationships each of which connects two people. A line represents a relationship between two people. Network A illustrates a network with 2 as the maximum degree of separation. Network B is disconnected.



Network A:
Max. degree of separation = 2



Network B:
Disconnected

Input

The input consists of data sets that describe networks of people. For each data set, the first line has two integers: P ($2 \le P \le 50$), the number of people in the network, and R ($R \ge 1$), the number of network relationships. Following that first line are R relationships. Each relationship consists of two strings that are names of people in the network who are related. Names are unique and contain no blank spaces. Because a person may be related to more than one other person, a name may appear multiple times in a data set.

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

Output

For each network, display the network number followed by the maximum degree of separation. If the network is disconnected, display `DISCONNECTED`. Display a blank line after the output for each network. Use the format illustrated in the sample output.

Sample Input	Output for the Sample Input
4 4 Ashok Kiyoshi Ursala Chun Ursala Kiyoshi Kiyoshi Chun 4 2 Ashok Chun Ursala Kiyoshi 6 5 Bubba Cooter Ashok Kiyoshi Ursala Chun Ursala Kiyoshi Kiyoshi Chun 0 0	Network 1: 2 Network 2: DISCONNECTED Network 3: DISCONNECTED

Arup's Steps

Filename: *steps*

For Christmas, Arup got a Fitbit. The only thing he likes more than taking lots of steps is looking at the data when he gets a chance to download it. Typically, on Fridays, after he parks, he walks in between several destinations: his car, his office, the Keurig machine, the HEC-308 lab where some COP 4516 students are and HEC-202, where the rest of the students are.

He'd like for you to make some pre-calculations for him, so that he doesn't have to go to the trouble of downloading data from his fitbit.

The Problem

Given the number of steps between various destinations as well as his path for a morning, determine the number of steps Arup has walked.

The Input

The first line of input will contain a single positive integer, n ($n \leq 26$), representing the number of locations Arup visits on morning walks. The locations will be lettered from 'A' to the n^{th} uppercase letter in the alphabet.

The next n lines will each contain n space separated integers. The first of these lines will contain the number of steps Arup takes to travel from the location lettered 'A' to each of the different locations in alphabetic order. The second of these lines will contain the number of steps Arup takes to travel from the location lettered 'B' to each of the different locations, in alphabetic order, and so on. The number of steps from a location to itself will always be 0, and the number of steps between any pair of distinct locations will be a positive integer less than or equal to 10,000. It is possible that the number of steps it takes Arup to go from location x to location y is different than the number of steps it takes him to go from location y to location x, for any distinct locations x and y.

The following line will contain a single positive integer, t ($t \leq 1000$), representing the number of morning walks of Arup's you have to evaluate. The following t lines each contain a single string of uppercase letters describing Arup's morning walk for that input case. The first letter of each string indicates the starting position for the walk and each subsequent letter represents the next location visited on the walk. The last letter of each string represents Arup's ending location for the walk. For example, if the input string is ACBCE, then this indicates that Arup starts at location A, walks directly to location C, then visits location B, followed by location C and then ends at location E. No other intermediate locations are visited, even if they save him steps. Each of these strings will contain in between 2 and 100 characters, inclusive and not contain the same letter appearing twice in a row.

The Output

For each test case, output a single integer representing the number of steps Arup took on the morning described in the test case.

Sample Input

```
5
0 750 825 649 720
730 0 125 200 53
807 125 0 100 127
649 185 100 0 139
720 53 127 139 0
3
ABCDEDBDE
ADED
AB
```

Sample Output

```
1777
927
750
```