Second Annual
University of Central Florida

High School Programming Tournament: Online Edition

Problems

Problem Name	Filename
Coloring Along the Edges	coloring
Eliminating Elbow Macaroni	elbow
Enraged Fowl	fowl
Text Graphics	graphics
Hangman	hangman
Jiminy's Jacuzzis	jacuzzi
The Smell of Adventure	smell
Series of Tubes	tubes

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

For example, if you are solving The Smell of Adventure:
Call your program file: smell.c, smell.cpp or smell.java
Call your Java class: smell

Coloring Along the Edges

Filename: `coloring`

Arthur the alien loves his alien-style connect-the-dots and always enjoys doing the connect-the-dots in his activity book (alien-style connect-the-dots is similar to the one you know but slightly different in that any pair of dots can be connected, not just consecutively-numbered ones). He also enjoys coloring his pictures and can color inside the edges, but sometimes he likes to change it up a bit and color along the edges. One of his favorite ways of coloring is to color the lines such that no two same-colored lines are touching each other at a dot. Unfortunately, he forgot most of his crayons, and only has two colors. He wants to know if the connect-the-dots he's working on can still be colored the way he wants them to be.

The Problem:

Given the configuration of the dots and the other dots to which each is connected, write a program that can determine whether or not it can be colored in the manner Arthur wants. Because he likes to do multiple activity book pages, he wants your program to work for multiple sets of dots and lines.

The Input:

Each input will begin with a line containing a positive integer, d, and a non-negative integer, n, where d is the number of dots and n is the number of lines ($d \leq 10$, $0 \leq n \leq d(d\text{-}1)/2$). For the next n lines, each line will contain two integers, x and y, between 1 and d, inclusive. These integers represent a line being drawn between the two dots x and y. Of course, a line between x and y is also a line between y and x. All lines between dots will be unique. The last line of input will be a line containing only "`0 0`". This line should not be processed.

The Output:

For each configuration of dots, output a line that has "`Page #i:`" where *i* is the number of the activity page (starting from 1). Then, on a line by itself output "`Yes, Arthur can color it.`" if the configuration is colorable by Arthur's method, or "`No, Arthur can't color it.`" if not. Leave a blank line after the output for each page.

Sample Input:

```
4 3
1 2
2 3
3 4
3 3
1 2
2 3
3 1
0 0
```

Sample Output:

```
Page #1:
Yes, Arthur can color it.

Page #2:
No, Arthur can't color it.
```

Eliminating Elbow Macaroni

Filename: `elbow`

Pete likes pasta, except for elbow macaroni. He's a little paranoid about it, since something might be hiding in the hollow part. In fact, he doesn't understand why macaroni is so popular, when there are plenty of pleasing pasta alternatives—including spaghetti, fettucini, rotini, and even ziti. (Ziti is also hollow, but it's straight, so you can see all the way through it!) Unfortunately for Pete, the school cafeteria often puts elbow macaroni in their soup. Pete likes the soup, but hates having to find and pick out all the elbow macaroni before he can eat it.

Fortunately, Pete has a robot with a 3-D backscatter X-ray scanner and a disintegrating laser. All he has to do is program the robot to look at his bowl of soup, find the bits of macaroni, and zap them with the laser. This might take a few seconds, but at least the laser zapping will keep the soup warm! The tricky part is recognizing elbow macaroni within the soup.

The robot perceives its target as a 3-dimensional grid. Pete has calibrated the grid so that the width of a unit cube will match the width of a typical piece of elbow macaroni. As a result, each piece of elbow macaroni will register as 3 unit cubes, which are contiguously connected (connected at their faces) to form an "L" shape.

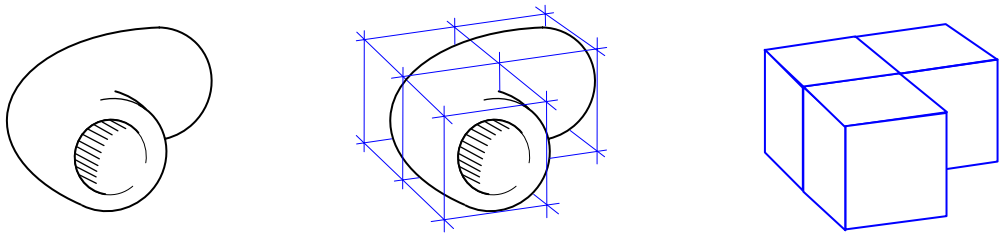


Figure 1. Robotic Digitization of Elbow Macaroni

The pernicious pasta might be floating in the soup in any orientation, but the robot will only be capable of finding pieces where it registers as an L-shape that is aligned to the grid axes. Pete is satisfied with this limitation, since the bizarre magnetic field of the school cafeteria causes 99.44% of all pasta to align to the grid axes anyway.

Pete doesn't want the robot to zap any tasty bits of beef, beans, or broccoli, even if they register as a shape that contains the characteristic L-shape of elbow macaroni. So the robot needs only to find L-shapes that are completely distinct from other shapes in the 3-D scan. Any contiguous group of cubes (connected at cube faces) that forms a T-shape, or an O-shape, or any other shape that contains the required L-shape, is presumed *not* to be elbow macaroni.

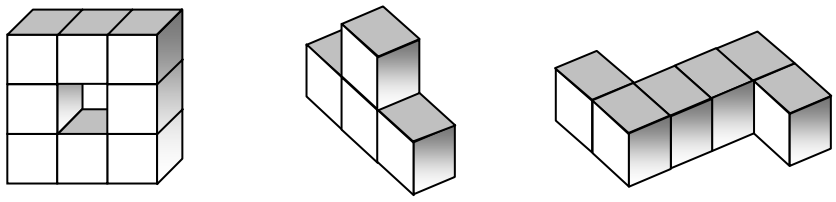


Figure 2. Some of the Shapes Presumed Not to be Elbow Macaroni

Cubes that touch only at an edge or corner are not considered to be contiguous.

The Problem:

Given a representation of the 3-dimensional grid perceived by the robot, find all L-shaped groups of 3 contiguous cubes that are distinct and separate from any other shapes, so that the robot can zap them out of the soup.

The Input:

The first line of input will contain only a positive integer, n, which is the number of scanned bowls that need to be evaluated and zapped. The representations for n scanned bowls will follow, starting on the next line.

The first line for each scanned bowl will contain only 3 integers, h ($2 \le h \le 40$), w ($4 \le w \le 100$), and b ($4 \le b \le 100$), separated by single spaces. These represent the height, width, and breadth of the bowl, respectively. (I don't know why, but the school uses rectangular bowls.) On the next $h \times w$ lines, there will be exactly b characters each, where each set of w lines of b characters represents a horizontal "slice," one unit cube's thickness, of the scanned bowl. These slices start from the top, with h=1.

Within the h^{th} such set of w lines, the b^{th} character on the w^{th} line represents the unit cube of the digitized grid that has coordinates (h, w, b). If the character is the uppercase letter 'O', it represents soup broth or something else the robot can ignore. If the character is the symbol '@', it is potentially a piece of pasta, but might be a bit of beef. No other characters will occur in the input.

The Output:

For each scanned bowl, output a line of the form

`Bowl #k:`

where k is the number of the bowl in the input, counting from 1.

Starting on the next line, output one line for each distinct L-shape found within the bowl, using the form

`Elbow macaroni found at (h,w,b). Zap it!`

where (h, w, b) are the coordinates of the corner piece of the L-shape. These messages should be ordered by ascending values for h, then w, then b.

If there are no distinct L-shapes found in the bowl, output the line

`No elbow macaroni found. What are they hiding now?`

Output a blank line after the message(s) for each bowl.

Sample Input:

```
2
2 8 12
OOOOOOOOOOOO
OO@@OOOOOOOO
OO@OOOOOO@OO
OOOOOOOOOOOO
OO@@@OOOOOOO
OO@O@OOOOOOO
OO@@@OOOOOOO
OOOOOOOO@@OO
@@@OOOOOOOOO
OOOOOOOOO@OO
OOOOOOOOO@OO
OOOO@@@OOOOO
OOOOO@OOOOOO
OOO@OOOO@OOO
OOOOOOOO@OOO
OOOOOOOO@OOO
3 4 6
@O@@OO
@O@O@O
OOOO@O
O@@OO@
OOO@@O
@O@O@O
@O@OOO
OO@OOO
OOOOOO
OOOOOO
@OO@OO
@OO@@@
```

Sample Output:

```
Bowl #1:
Elbow macaroni found at (1,2,3). Zap it!
Elbow macaroni found at (2,3,10). Zap it!

Bowl #2:
No elbow macaroni found. What are they hiding now?
```

Enraged Fowl

Filename: `fowl`

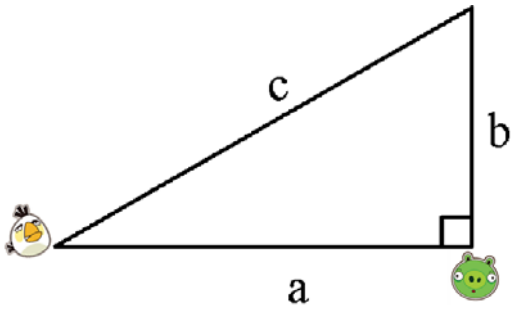
Brandon is playing the latest game sensation, Enraged Fowl! His favorite fowl to use in this game is the one that "flies" in a straight line and then drops a bomb straight down on a target. Unfortunately, he is not very good at dropping the bomb so he wants to practice outside of the game. He wants you 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
```

Text Graphics

Filename: `graphics`

You want to design a simple language that allows the creation of text graphics. You start with a blank canvas of a given size that holds text characters. Each command, executed in order, will allow you to paint any rectangular portion of the canvas with a particular character. There are two modes of painting that apply to an entire program: one where old characters get painted over (N), and another where new characters only appear on blank squares (B). The mode of a program is always specified at the beginning of the program. Your job is to execute these text graphics programs.

The canvas for each program will be given by a set of dimensions with *r* rows and *c* columns. These will be numbered from 0 to *r*-1 and 0 to *c*-1, respectively, with the top-left corner being row 0 column 0 and the bottom-right corner being row *r*-1 column *c*-1. The only valid command for the language will have the following form:

sr sc er ec ch

where *sr* and *sc* are the row and column, respectively, of the top left-hand corner of the rectangle to paint, *er* and *ec* are the bottom right-hand corner of the rectangle to paint, and *ch* is the character with which to paint the rectangle.

The Problem:

Given a text graphics program, show the fully painted canvas.

The Input:

The first line of the input file will contain a single positive integer, *n*, representing the number of programs to execute. The programs follow. The first line of each program will contain the number of rows, *r* ($0 < r \leq 50$), number of columns, *c* ($0 < c \leq 50$), and the mode of the program (either the character `'N'` or `'B'`), each separated by a single space. The following lines, in order, represent the lines of the program. Each valid line of the program will contain five items of information separated by spaces: s_r, s_c, e_r, e_c, ($s_r \leq e_r < r$, $s_c \leq e_c < c$), and *ch*, where s_r, and s_c are non-negative integers that represent the row and column of the upper left-hand corner of the block to draw, e_r and e_c are non-negative integers that represent the row and column of the bottom right-hand corner of the block to draw, and *ch* represents the character with which to fill each of those squares, in the given rectangle, inclusive. *ch* will be a printable ASCII character. The end of a single program is designated by the word "END" on a line by itself. The next program, if there is one, starts immediately on the next line.

The Output:

The first line of each canvas should begin with the header "`Canvas #`*k*" where *k* is the number of the canvas from the input, starting at 1. Following this, the first row of output should have one space followed by the unit's digit of the column number, starting at 0. Each subsequent row will have as its first character the unit's digit of the row number, followed by that row of the canvas, which contains exactly *c* characters (a blank position should be output as a space), of the final state of the canvas for each program, followed by the unit's digit of the row number at the end as well. Separate each output canvas with a blank line.

Sample Input:

```
2
10 20 N
3 5 7 15 #
3 2 9 7 &
END
5 10 B
0 0 4 5 ^
1 3 3 8 )
END
```

Sample Output:

```
Canvas #1
 01234567890123456789
0                    0
1                    1
2                    2
3  &&&&&&########    3
4  &&&&&&########    4
5  &&&&&&########    5
6  &&&&&&########    6
7  &&&&&&########    7
8  &&&&&&            8
9  &&&&&&            9

Canvas #2
 0123456789
0^^^^^^    0
1^^^^^^))) 1
2^^^^^^))) 2
3^^^^^^))) 3
4^^^^^^    4
```

Hangman

Filename: `hangman`

Ali was recently captured by his evil twin arch-nemesis, Ila, and there is only one way for Ali to save himself: playing Ila's favorite game... Hangman. Ila has placed Ali over a pit of lava, and set a key-phrase to close the lava pit door. Like all evil masterminds, Ila leaves Ali after setting the key-phrase and just assumes that he will be melted like a grilled cheese sandwich. Ali plays the game by yelling out letters, in any order. The computer fills all occurrences of those letters into the key-phrase if they exist; otherwise it makes the rope from which he is hanging drop one foot. Ali knows that if he just sits there, he will never escape, so he has to shout out letters. If a shouted letter is contained in the key-phrase, then he doesn't get lowered closer to the lava. If he shouts a letter he already shouted (whether it is in the key phrase or not; remember, he is panicking!), or if the letter he shouts is not part of the key phrase, then he gets lowered 1 foot closer to the lava. If Ali ever touches the lava (i.e., his height becomes zero feet above the lava), he becomes extra crispy. However, if Ali ever completes the key phrase, then the lava pit will be closed and Ali will be freed from his bindings!

The Problem:

Given the height above the lava Ali is hanging (in feet), a string giving the required key phrase, and another string giving the order in which Ali will be guessing letters, determine whether Ali will escape or burn in the lava like a grilled cheese sandwich!

The Input:

The first line of the input will contain only a single positive integer, t, the number of "hang-man-over-lava-pit" games to be checked. Each of the following t lines will contain an integer, n ($1 \leq n \leq 15$), representing the height above the lava that Ali is hanging (in feet), the key-phrase of length m ($1 \leq m \leq 100$) to close the lava pit, and a string of length r ($n \leq r \leq 100$) with the letters that Ali will call out in the order he calls them. Only lowercase letters are used. Each element will be separated from the others by a single space. Ali will always call out enough letters to finish the game – one way or the other. He doesn't want to be stuck hanging forever!

The Output:

For each test case output one line, starting with "`Lava pit #`*u*`: `" where *u* is the current game being processed, starting at 1. Then, if Ali will live, print "`Ali survives!`" Otherwise, output "`Poor Ali.`"

Sample Input:

```
2
5 password arsdqewpttosdgs
3 password qweapsorjdyui
```

Sample Output:

```
Lava pit #1: Ali survives!
Lava pit #2: Poor Ali.
```

Jiminy's Jacuzzis

Filename: `jacuzzi`

Jiminy is opening up a new Jacuzzi business. In order to set-up his showroom, he had his assistant assemble each model of Jacuzzi. Unfortunately, his assistant made a poor choice and mixed up all the sides of all of the Jacuzzis! Now, Jiminy and his assistant are having trouble piecing sides together in order to build the Jacuzzis. They need your help!

The Problem:

Given the lengths of the sides that may make up a Jacuzzi, determine whether it is possible to build one using all of the sides or not. A Jacuzzi is formed by aligning the sides into a polygonal shape with positive area (e.g. the outline of the Jacuzzi). Note that neither Jiminy nor his assistant can cut any of the sides (they must use them as-is) and all sides must be used when constructing each Jacuzzi.

The Input:

The input will begin with a line containing a positive integer, t, representing the number of Jacuzzis to check. Each Jacuzzi then starts with a line containing an integer, n ($1 \leq n \leq 100$), representing the number of sides of the Jacuzzi being built. The next line contains n integers representing the length of each side; each integer will be separated by a single space. All of the side lengths are between 1 and 100, inclusive.

The Output:

For each Jacuzzi, output a line "`Jacuzzi #`*x*`: `*m*" where x is the number of the Jacuzzi in the input (starting from 1) and m is "`YES`" if Jiminy can build the Jacuzzi or "`NO`" otherwise.

Sample Input:

```
5
5
1 2 3 4 11
3
1 2 4
6
2 3 4 5 6 7
3
1 1 10
6
99 2 3 4 5 6
```

Sample Output:

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

The Smell of Adventure

Filename: `smell`

Hello, programmers. How are you? Fantastic! As you know, I'm the Man Your Man Could Smell Like, and I have a mission for you. I have been making an effort to inform the people on this crazy blue marble that we call Earth about the smell of adventure. Although my commercials have had great success, I'm looking to expand my work to new precipices in the form of an application for your mobile cellular device. Part of this application involves a user entering in the names of friends, family members, loved ones, pets, and strangers to determine if they do indeed like the smell of adventure. But, as I'm sure you know, everyone likes the smell of adventure, so you need merely respond in the affirmative.

The Problem:

You will be given a name, which is a sequence of between 1 and 100 (inclusive) upper and lower case letters. You are to take that name and output whether they like the smell of adventure (but, of course, they do!). Names will never include numbers, white space, or any symbol other than letters. Your application should be able to do this task repeatedly since a user may want to check more than one person's like of the smell of adventure.

The Input:

You will be given an integer, *t*, which is the number of names you will test. The following *t* lines will each contain a single name on its own line, as formatted above. Names will never have leading or trailing whitespace, except for the single newline character separating them.

The Output:

For each of the *t* names, output "`Name #i:`" (where *i* is the number of the current name, starting at 1) on a line by itself. Then output "`Does n like the smell of adventure? Of course they do.`" (where *n* is the name read in, capitalized exactly as it was given to you). After that, print a blank line to separate each such output.

Sample Input:

```
3
Ali
Josh
JimmyFlowers
```

Sample Output:

```
Name #1:
Does Ali like the smell of adventure? Of course they do.

Name #2:
Does Josh like the smell of adventure? Of course they do.

Name #3:
Does JimmyFlowers like the smell of adventure? Of course they do.
```

Series of Tubes

Filename: `tubes`

Redwood: Hello there! Welcome to the world of the INTERNETZ! My name is REDWOOD! People call me the INTERNETZ PROF!

Redwood: This world is inhabited by creatures called LOLCATZ! For some people LOLCATZ are pets. Others use them for sending messages. Myself...I study LOLCATZ as a profession.

Redwood: As you already know, the INTERNETZ is just a series of tubes. People on the INTERNETZ use LOLCATZ to communicate with one another. The tubes have all different sizes and sometimes LOLCATZ are too big to go through the tubes. When the LOLCATZ can't get to their destination they get stuck in the INTERNETZ and clog the tubes! OH NOES!

Redwood: I have a request for you. On the desk there is my invention, the LOLDEX! It allows you to directly communicate with the INTERNETZ and find out the size of each tube. Using it I would like you to determine the biggest possible LOLCAT that I can send to each computer without causing a clog in the INTERNETZ

Redwood: Your very own INTERNETZ legend is about to unfold! A world of dreams and adventures with LOLCATZ awaits! Let's go!

The Problem:

Given a series of tubes from the LOLDEX and their sizes, determine the largest LOLCAT that can make it to each computer from the LOLDEX. A LOLCAT can get to a computer if there exists a path through the INTERNETZ such that the LOLCAT does not get stuck! If the LOLCAT is at most the same size as a tube, then the LOLCAT will not get stuck in that tube. There will never be more than one tube from one computer to another. LOLCATZ can go either direction in a tube.

(Problem Statement continues on next page)

The Input:

There will be multiple series of tubes to process, each representing the INTERNETZ at some point in time. Each series of tubes starts with a positive integer, n ($1 \leq n \leq 50$), representing the number of computers connected to the INTERNETZ. On the same line after a space is a number, t ($1 \leq t \leq 150$), representing the number of tubes that the LOLDEX provides. The following t lines contain only integers, i, j and s ($0 \leq i < j < n$; $1 \leq s \leq 50$), separated by a single space where each represents a tube of size s between computer i and computer j. The LOLDEX is represented as the 0-indexed computer in the INTERNETZ. The last series is followed by two zeros (separated by a single space), which indicates the end of input and should not be processed.

The Output:

For each series, output "`Series of tubes #`*t*`:`" where t is the number of the current series being processed (beginning with 1). Follow this with lines outputting the maximum size LOLCAT that can reach each computer from the LOLDEX in the form "`Computer `*c*` can receive up to a size `*s*` LOLCAT.`" However, if a LOLCAT cannot reach Computer c, skip processing that computer. The computers are to be output in order from Computer 1 to Computer n-1 and you are guaranteed that at least one computer will be reachable by a LOLCAT. Do not output the LOLDEX since it can handle any size LOLCAT. Follow each series with a single blank line.

Sample Input:

```
3 3
0 1 2
0 2 4
1 2 3
5 5
0 1 1
1 2 16
0 2 8
0 4 50
2 4 1
0 0
```

Sample Output:

```
Series of tubes #1:
Computer 1 can receive up to a size 3 LOLCAT.
Computer 2 can receive up to a size 4 LOLCAT.

Series of tubes #2:
Computer 1 can receive up to a size 8 LOLCAT.
Computer 2 can receive up to a size 8 LOLCAT.
Computer 4 can receive up to a size 50 LOLCAT.
```