2009 Stanford Local ACM Programming Contest

Saturday, October 3rd, 2009

Read these guidelines carefully!

Rules

1. You may use resource materials such as books, manuals, and program listings. You may *not* search for solutions to specifics problems on the Internet, though you are permitted to use online language references (e.g., the Java API documentation) or algorithms references equivalent to what would be found in a standard algorithms textbook (e.g., CLRS). You may *not* use any machine-readable versions of software, data, or existing electronic code. That is, all programs submitted *must be typed during the contest*. No cutting and pasting of code is allowed.

2. You may not collaborate in any way (verbally, electronically, in writing, using gestures, telepathically, etc.) with other contestants, students, or anyone else during the contest.

3. You are expected to adhere to the honor code. You are still expected to conduct yourself according to the rules, even if you are not participating on site in the Gates building.

Guidelines for submitted programs

1. All programs must be written in C, C++, or Java. For judging, we will compile the programs in the following way:

   - `.c`: using `gcc -O2 -lm` (GCC version 4.2.4)
   - `.cc`: using `g++ -O2 -lm` (GCC version 4.2.4)
   - `.java`: using `javac` (Sun Java version 1.5.0)

   All programs will be compiled and tested on a Leland `myth` machine. The `myth` machines are dual Xeon 3.20 GHz machines with 2 GB RAM running Ubuntu Linux 7.10. Compilation errors or other errors due to incompatibility between your code and the `myth` machines will result in a submission being counted incorrect.

2. Make sure you `return 0;` in your `main()`; **any non-zero return values may be interpreted by the automatic judge as a runtime error**.

3. **Java users:** Please place your `public static void main()` function in a public class with the same name as the base filename for the problem. For example, a Java solution for the `test` program should be submitted in the file `test.java` and should contain a `main()` in `public class test`.

4. All solutions must be submitted as a single file.

5. All programs should accept their input on **stdin** and produce their output on **stdout**. They should be batch programs in the sense that they do not require human input other than what is piped into stdin.

6. Be sure to follow the output format described in the problem exactly. We will be judging programs based on a `diff` of your output with the correct solution, so your program's output must match the judge output **exactly** for you to receive credit for a problem. As a note, each line of an output file must end in a newline character, and there should be no trailing whitespace at the ends of lines.

How will the contest work?

1. If you chose to work remotely from a home computer, we recommend that you test out your account on the online contest system by submitting a solution for the test problem shown on the next page. We will do our best to set up the contest host to accept test problem submissions Saturday morning until approximately 1:45 pm.

2. For those who choose to participate onsite, from 1:00 to 1:45 pm, you should select a computer (or find a place to plug in your laptop), set up your workspace and complete a test problem. Space in Gates B02 and the PUP cluster is limited, and will be available on a first-come first-served basis. You may also choose to work directly on one of the `myth` machines in Gates B08, although technically we do not have that room reserved for the contest.

3. At 2:00 pm, the problems will be posted on the live contest page in PDF format, all registered participants will be sent an e-mail that the problems have been posted, and we will distribute paper copies of the problems to contestants competing in either Gates B02 or the PUP cluster.

4. For every run, your solution will be compiled, tested, and accepted or rejected for one of the following reasons: *compile error*, *run-time error*, *time limit exceeded*, *incorrect output*, or *presentation error*. In order to be accepted, your solution must match the judge output exactly (according to `diff`) on a set of hidden judge test cases, which will be revealed after the contest.

    - Source code for which the compiler returns errors (warnings are ok) will be judged as *compile error*.
    - A program which returns any non-zero error code will be judged as *run-time error*.
    - A program which exceeds the time allowed for any particular problem will be judged as *time-limit exceeded* (see below).
    - A program which fails a `diff -w -B` will be judged as *incorrect output*.
    - A program which passes a `diff -w -B` but fails a `diff` (i.e., output matches only when ignoring whitespace and blank lines) will be judged as *presentation error*.
    - A program which passes a `diff` and runs under the time constraints specified will be judged as *accepted*.

5. For each problem, the time allowed for a run (consisting of multiple test cases) will be 10 seconds total for all test cases. The number of test cases in a run may vary from 20 to 200 depending upon the problem, so be sure to write algorithmically efficient code!

6. You can view the status of each of your runs on the live online contest site. Please allow a few minutes for your submissions to be judged. The site also provides a live scoreboard for you to watch the progress of the contest as it unfolds.

7. At 6:00 pm, the contest will end. No more submissions will be accepted. Contestants will be ranked by the number of solved problems. Ties will be broken based on total time, which is the sum of the times for correct solutions; the time for a correct solution is equal to the number of minutes elapsed since 2:00 pm plus 20 penalty minutes per rejected solution. No penalty minutes are charged for a problem unless a correct solution is submitted. After a correct submission for a problem is received, all subsequent incorrect submissions for that problem do not count towards the total time.

8. The results of this contest will be used in part to select team members for representing Stanford at the forthcoming ACM regional competition. Six or more contestants have been customarily invited to the Stanford ACM ICPC teams in previous years.

Helpful hints

1. **Make sure your programs compile and run properly on the `myth` machines.** If you choose not to develop on the Leland systems, you are responsible for making sure that your code is portable.

2. **Read (or skim) through all of the problems at the beginning to find the ones that you can code quickly.** Finishing easy problems at the beginning of the contest is especially important as the time for each solved problem is measured from the beginning of the contest. Also, check the leaderboard frequently in order to see what problems other people have successfully solved in order to get an idea of which problems might be easy and which ones are likely hard.

3. If you are using C++ and unable to get your programs to compile/run properly, try adding the following line to your `.cshrc` file

   ```
   setenv LD_LIBRARY_PATH /usr/pubsw/lib
   ```

   and re-login.

4. The `myth` machines in Gates B08 are not officially reserved for the contest, but these will be the machines used for judging/testing of all programs. You may find it helpful to work on these machines in order to ensure compatibility of your code with the judging system.

5. If you are a CS major and have a working `xenon` account, please work in the **PUP cluster** rather than Gates B08; the PUP cluster has really nice Linux machines with very little load. (Do check the Java configuration if that affects you though!) If you don't know where the PUP cluster is, just ask!

6. If you are working on your own (Windows) laptop in Gates B02, but plan to ssh into a myth machine, it may be helpful to run a VNC session. Check out the IT services page on using VNC, which can be found at `http://unixdocs.stanford.edu/moreX.html`. If you wish to use an IDE (e.g., Visual Studio or Eclipse), please make sure that you know how to set this up yourself beforehand. We will not be able to provide technical support related to setting up IDEs during the contest.

7. If you need a clarification on a problem or have any other questions, post an clarification request to the live contest page, or just come talk to us. The contest judges will likely be hiding in the myth cluster, **Gates B08**.

The directions given here are originally based on those taken from Brian Cooper's 2001 Stanford Local Programming Contest problem set, and have been updated year after year to the best of our ability. The contest organizers would like to thank the problem authors of 2009, in alphabetical order, Andy Nguyen, Chuong Do, Jonathan Lee, and Sonny Chan.

0 Test Problem (`test.{c,cc,java}`)

0.1 Description

This is a test problem to make sure you can compile and submit programs. Your program for this section will take as input a single number N and return the average of all integers from 1 to N, inclusive.

To submit a solution, navigate your browser to the live contest page, which this year resides at:

`http://cs.stanford.edu/group/acm/slpclive/`

You must log in using your assigned user ID and password for the contest. If you have registered for the contest, but have not received an email message containing your login information, then it's time to contact one of the contest organizers in panic! Students in Gates B02 or the PUP cluster will also be given their login information on paper before the contest.

Use the "Submissions" tab on the contest page to submit your solution to the problem. Detailed instructions can be found in the "Help" section online if necessary.

After submitting your solution, please allow a few minutes for it to be judged. You can resubmit rejected solutions as many times as you like (though incurring a 20 minute penalty for each rejected run of a problem you eventually get right). Once you have submitted a correct solution, future submissions of that problem will still be graded but will not count towards your final score or total time.

Note that you do not need to submit this problem during the actual contest!

0.2 Input

The input test file will contain multiple test cases. Each test case is specified on a single line containing an integer N, where $-100 \leq N \leq 100$. The end-of-file is marked by a test case with $N = -999$ and should not be processed. For example:

```
5
-5
-999
```

0.3 Output

The program should output a single line for each test case containing the average of the integers from 1 to N, inclusive. You should print numbers with exactly two decimal places. For example:

```
3.00
-2.00
```

0.4 Sample C Solution

```
#include <stdio.h>

int main() {
  int n;
  while (1) {
    scanf("%d", &n);
    if (n == -999) break;
    if (n > 0) printf("%.2lf\n", (double)(n * (n + 1) / 2) / n);
    else printf("%.2lf\n", (double)(1 + n * (1 - n) / 2) / (2 - n));
  }
  return 0;
}
```

0.5 Sample C++ Solution

```
#include <iostream>
#include <iomanip>
using namespace std;

int main() {
  cout << setprecision(2) << setiosflags(ios::fixed | ios::showpoint);
  while (true) {
    int n;
    cin >> n;
    if (n == -999) break;
    if (n > 0) cout << double(n * (n + 1) / 2) / n << endl;
    else cout << double(1 + n * (1 - n) / 2) / (2 - n) << endl;
  }
  return 0;
}
```

0.6 Sample Java Solution

```
import java.util.*;
import java.text.DecimalFormat;

public class test {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        DecimalFormat fmt = new DecimalFormat("0.00");
        while (true) {
            int n = s.nextInt();
            if (n == -999) break;
            if (n > 0) System.out.println(fmt.format((double)(n * (n + 1) / 2) / n));
            else System.out.println(fmt.format((double)(1 + n * (1 - n) / 2) / (2 - n)));
        }
    }
}
```

A Bonsai (`bonsai.{c,cc,java}`)

A.1 Description

After being assaulted in the parking lot by Mr. Miyagi following the "All Valley Karate Tournament", John Kreese has come to you for assistance. Help John in his quest for justice by chopping off all the leaves from Mr. Miyagi's bonsai tree!

You are given an undirected tree (i.e., a connected graph with no cycles), where each edge (i.e., branch) has a nonnegative weight (i.e., thickness). One vertex of the tree has been designated the root of the tree. The remaining vertices of the tree each have unique paths to the root; non-root vertices which are not the successors of any other vertex on a path to the root are known as leaves.

Determine the minimum weight set of edges that must be removed so that none of the leaves in the original tree are connected by some path to the root.

A.2 Input

The input file will contain multiple test cases. Each test case will begin with a line containing a pair of integers n (where $1 \leq n \leq 1000$) and r (where $r \in \{1, \ldots, n\}$) indicating the number of vertices in the tree and the index of the root vertex, respectively. The next $n-1$ lines each contain three integers u_i v_i w_i (where $u_i, v_i \in \{1, \ldots, n\}$ and $0 \leq w_i \leq 1000$) indicating that vertex u_i is connected to vertex v_i by an undirected edge with weight w_i. The input file will not contain duplicate edges. The end-of-file is denoted by a single line containing "0 0".

```
15 15
1 2 1
2 3 2
2 5 3
5 6 7
4 6 5
6 7 4
5 15 6
15 10 11
10 13 5
13 14 4
12 13 3
9 10 8
8 9 2
9 11 3
0 0
```

A.3 Output

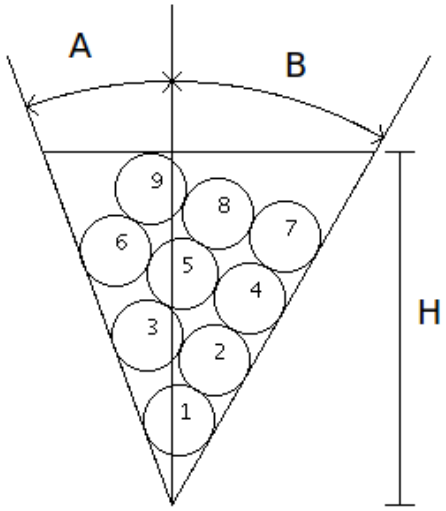
For each input test case, print a single integer indicating the minimum total weight of edges that must be deleted in order to ensure that there exists no path from one of the original leaves to the root.

```
16
```

B Fruit Bowl (`fruit.{c,cc,java}`)

B.1 Description

For her next project, Beebe Flat wishes to paint a bowl of fruit. Naturally, the bowl is two-dimensional, as is all of the fruit. She bought a triangular bowl which somehow always manages to stay upright, with a perfectly level top. The bowl has a height H, and opens A degrees to the left and B degrees to the right (see diagram). In the interest of art, Beebe eschews symmetry, so $A \neq B$. She plans to buy perfectly circular fruit, each with radius 1, to put into the bowl. However, perfectly circular fruit is expensive, so she needs your help to figure out how much fruit she has to buy. She plans to fill the bowl to the brim, adding as much fruit as possible without any part exceeding the height of the bowl. Thankfully, she is not interested in an optimal packing, since she wants a simple algorithm for actually arranging the fruit. She will place the fruit one at a time, each time choosing the lowest possible location for the center.



Since Beebe doesn't like to deal with ties, she always purchases a bowl such that there is only one lowest possible location within a margin of 10^{-5}. Finally, to ensure that a lid will fit nicely on the bowl when she's done painting, she only chooses bowls such that, when properly packed, the last piece of fruit to fit will be at least 10^{-2} below the top, and the next piece of fruit that would fit if the bowl were taller will jut out at least 10^{-2} above the top.

Given a particular bowl, how many pieces of fruit does Beebe need to buy to fill the bowl in this manner?

B.2 Input

The input consists of multiple test cases. Each test case has three integers on a single line, denoting A, B, and H. $1 \leq A, B \leq 45$, $A \neq B$, and $1 \leq H \leq 300$. The last test case will be followed by $A = B = H = 0$, which should not be processed. For example:

```
20 30 10
10 20 20
0 0 0
```

B.3 Output

For each test case, print on a single line the number of pieces of fruit that Beebe needs to buy to fill the given bowl. For example:

```
9
25
```

C Number Game (`game.{c,cc,java}`)

C.1 Description

Carl and Ellie are in the midst of another adventure; this time, it is a road trip through Canada! They've just arrived in Saskatchewan [insert rectangular shape here], right in the middle of the Canadian prairies, and, to their horror, have discovered that all the rumours about it being dreadfully flat are true. Suddenly, the warnings their Canadian friends gave them prior to the trip spring back to mind, such as:

> It's so flat, a boy can watch his dog run away!

Or:

> It's so flat, it's impossible to jump to your death!

As the driver, Carl is worried about falling asleep at the wheel and has decided to come up with a game to relieve the boredom. As a conscientious driver, he doesn't want the game to be too distracting—the roads here are unerringly straight, so it's easy to lose track of motion—so the rules are simple:

- Carl picks a number, N, between 1 and $1,000,000,000$.
- Carl and Ellie take turns subtracting an integer (between 1 and 20) from N. Carl plays first, and the winner is the one who subtracts off a number to get 0.

For example, suppose Carl picks the number 50. He subtracts off the number 5, leaving 45. Ellie subtracts off 17, leaving 28. Carl subtracts off 8, leaving 20. Finally, Ellie subtracts off 20, leaving 0, and wins!

Frankly, Ellie would rather sleep than play this game, so she has reprogrammed the GPS (which isn't necessary in this region, anyway) to play for her instead. Her method of choosing a number is straightforward:

- If on a given turn, the number remaining is 20 or less, then she picks that number and wins.
- Otherwise, her choice of number is completely determined by the number Carl just picked, as follows. Before the game starts, Ellie chooses 20 random numbers, $1 \leq a_1, a_2, a_3, \ldots, a_{20} \leq 20$. Then whenever Carl subtracts off the number k, Ellie responds by subtracting off the number a_k (unless she can win).

Carl needs your help! To help stay motivated playing this game, he would like to know if there exists a winning strategy for him, given the numbers N and $a_1, a_2, a_3, \ldots, a_{20}$.

C.2 Input

Each input case begins with the number N on a line by itself; the next line contains the numbers $a_1, a_2, \ldots, a_{20}$, separated by spaces. Input terminates with a line containing 0.

```
42
20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
0
```

C.3 Output

For each test case, print "Carl can win" or "Carl can't win".

```
Carl can't win
```

In this example, since Carl chooses N to be 42, it turns out that he has no winning strategy. (Indeed, whichever number he initially subtracts off, Ellie will subtract off a number leaving 21 for Carl. Then whatever number Carl subtracts off next, Ellie will be left with a number between 1 and 20, and she wins!)

D Leap Frog (`leapfrog.{c,cc,java}`)

D.1 Description

Jack and Jill play a game called "Leap Frog" in which they alternate turns jumping over each other. Both Jack and Jill can jump a maximum horizontal distance of 10 units in any single jump. You are given a list of valid positions $x_1, x_2, \ldots, x_n$ where Jack or Jill may stand. Jill initially starts at position x_1, Jack initially starts at position x_2, and their goal is to reach position x_n.

Determine the minimum number of jumps needed until either Jack or Jill reaches the goal. The two players are never allowed to stand at the same position at the same time, and for each jump, the player in the rear must hop over the player in the front.

D.2 Input

The input file will contain multiple test cases. Each test case will begin with a single line containing a single integer n (where $2 \leq n \leq 100000$). The next line will contain a list of integers $x_1\ x_2\ \ldots\ x_n$ where $0 \leq x_1 < x_2 \ldots < x_n \leq 1000000$. The end-of-file is denoted by a single line containing "0".

```
6
3 5 9 12 15 17
6
3 5 9 12 30 40
```

D.3 Output

For each input test case, print the minimum total number of jumps needed for both players such that either Jack or Jill reaches the destination, or -1 if neither can reach the destination.

```
3
-1
```

E Universal Oracle (`oracle.{c,cc,java}`)

E.1 Description

In computer science, an oracle is something that gives you the answer to a particular question. For this problem, you need to write an oracle that gives the answer to everything. But it's not as bad as it sounds; you know that 42 is the answer to life, the universe, and everything.

E.2 Input

The input consists of a single line of text with at most 1000 characters. This text will contain only well-formed English sentences. The only characters that will be found in the text are uppercase and lowercase letters, spaces, hyphens, apostrophes, commas, semicolons, periods, and question marks. Furthermore, each sentence begins with a single uppercase letter and ends with either a period or a question mark. Besides these locations, no other uppercase letters, periods, or question marks will appear in the sentence. Finally, every question (that is, a sentence that ends with a question mark) will begin with the phrase "What is..." For example:

```
Let me ask you two questions. What is the answer to life? What is the answer to the universe?
```

E.3 Output

For each question, print the answer, which replaces the "What" at the beginning with "Forty-two" and the question mark at the end with a period. Each answer should reside on its own line. For example:

```
Forty-two is the answer to life.
Forty-two is the answer to the universe.
```

F Rectangles (`rectangles.{c,cc,java}`)

F.1 Description

A rectangle in the Cartesian plane is specified by a pair of coordinates (x_1, y_1) and (x_2, y_2) indicating its lower-left and upper-right corners, respectively (where $x_1 \leq x_2$ and $y_1 \leq y_2$). Given a pair of rectangles, $A = ((x_1^A, y_1^A), (x_2^A, y_2^A))$ and $B = ((x_1^B, y_1^B), (x_2^B, y_2^B))$, we write $A \preceq B$ (i.e., A "precedes" B), if

$$x_2^A < x_1^B \qquad \text{and} \qquad y_2^A < y_1^B.$$

In this problem, you are given a collection of rectangles located in the two-dimension Euclidean plane. Find the length L of the longest sequence of rectangles $(A_1, A_2, \ldots, A_L)$ from this collection such that

$$A_1 \preceq A_2 \preceq \ldots \preceq A_L.$$

F.2 Input

The input file will contain multiple test cases. Each test case will begin with a line containing a single integer n (where $1 \leq n \leq 1000$), indicating the number of input rectangles. The next n lines each contain four integers x_1^i y_1^i x_2^i y_2^i (where $-1000000 \leq x_1^i \leq x_2^i \leq 1000000$, $-1000000 \leq y_1^i \leq y_2^i \leq 1000000$, and $1 \leq i \leq n$), indicating the lower left and upper right corners of a rectangle. The end-of-file is denoted by a single line containing the integer 0.

```
3
1 5 2 8
3 -1 5 4
10 10 20 20
2
2 1 4 5
6 5 8 10
0
```

F.3 Output

For each input test case, print a single integer indicating the length of the longest chain.

```
2
1
```

G Rectangles Too! (`rectanglestoo.{c,cc,java}`)

Note: This problem is almost identical to the previous problem. The single difference between the two problems has been marked below.

G.1 Description

A rectangle in the Cartesian plane is specified by a pair of coordinates (x_1, y_1) and (x_2, y_2) indicating its lower-left and upper-right corners, respectively (where $x_1 \leq x_2$ and $y_1 \leq y_2$). Given a pair of rectangles, $A = ((x_1^A, y_1^A), (x_2^A, y_2^A))$ and $B = ((x_1^B, y_1^B), (x_2^B, y_2^B))$, we write $A \preceq B$ (i.e., A "precedes" B), if

$$x_2^A < x_1^B \qquad \text{and} \qquad y_2^A < y_1^B.$$

In this problem, you are given a collection of rectangles located in the two-dimension Euclidean plane. Find the length L of the longest sequence of rectangles $(A_1, A_2, \ldots, A_L)$ from this collection such that

$$A_1 \preceq A_2 \preceq \ldots \preceq A_L.$$

G.2 Input

The input file will contain multiple test cases. Each test case will begin with a line containing a single integer n (where $1 \leq n \leq$ **100000**), indicating the number of input rectangles. The next n lines each contain four integers x_1^i y_1^i x_2^i y_2^i (where $-1000000 \leq x_1^i \leq x_2^i \leq 1000000$, $-1000000 \leq y_1^i \leq y_2^i \leq 1000000$, and $1 \leq i \leq n$), indicating the lower left and upper right corners of a rectangle. The end-of-file is denoted by a single line containing the integer 0.

```
3
1 5 2 8
3 -1 5 4
10 10 20 20
2
2 1 4 5
6 5 8 10
0
```

G.3 Output

For each input test case, print a single integer indicating the length of the longest chain.

```
2
1
```

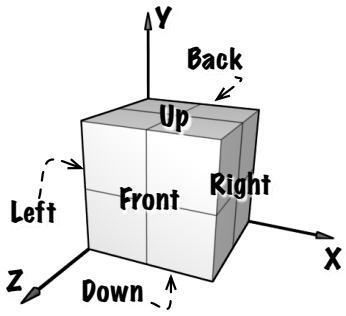
H Rubik 2^3 (`rubik2.{c,cc,java}`)

H.1 Description

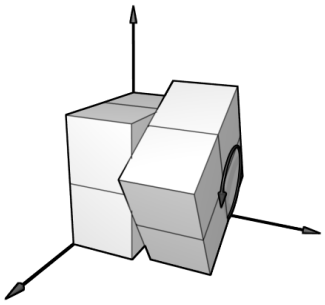
Sonny is probably the only computer science Ph.D. student who cannot solve a Rubik's cube. One day, he came across a neat little $2 \times 2 \times 2$ Rubik's cube, and thought, "Finally, here's a cube that's easy enough for me to do!" Nope, wrong! He got pwned, hardcore. How embarrassing.

To ensure that this does not happen again, he decides to write a computer program to solve the cube. Then he had this brilliant idea: Why not have the students at the programming contest do the work instead? So, given an initial configuration of the $2 \times 2 \times 2$ Rubik's cube, your task for this problem is to write a program that solves it.

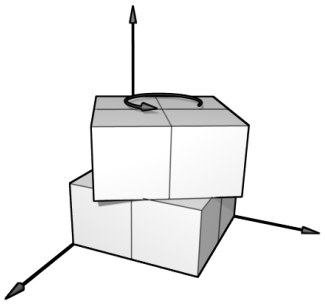
The mini-cube has 6 faces, each with 4 painted tiles on it. The faces are labeled Front (F), Back (B), Up (U), Down (D), Left (L), and Right (R), according to the diagram below. Each of the tiles on the faces can be colored Red (R), Green (G), Blue (B), Yellow (Y), Orange (O), or White (W), and there are exactly 4 instances of each color. The cube is considered solved when the colors of all tiles on each distinct face of the cube match.



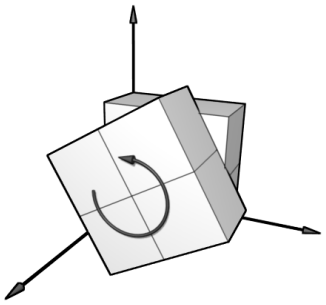
You may use any combination of three distinct moves to transform the cube: a turn about the X-axis, a turn about the Y-axis, or a turn about the Z-axis. Each turn is exactly 90 degrees of all tiles on half the cube, in the directions illustrated below. Note that the back-down-left corner is fixed with respect to all valid transforms.



X-axis turn



Y-axis turn



Z-axis turn

Can you come up with a sequence of moves that will solve a given configuration of the Rubik's cube?

H.2 Input

You will be given maps of an "unwrapped" cubes showing colors on each of the faces, in the following format:

```
..UU....
..UU....
LLFFRRBB
LLFFRRBB
..DD....
..DD....
```

The letters in the above diagram shows you where to find the colors on each face (as shown in the first diagram) from the map only – it is not valid input! The front face is oriented as in the diagram, with the other faces on the map attached to it so that it wraps to cover the cube. The letters on the faces may be any of R, G, B, Y, O, or W to indicate the color. Dot (.) characters serve to pad the map to a 6×8 grid, and are of no other significance.

The input consists of several configuration maps in the format described, separated by blank lines. You may assume that each configuration is both valid and solvable. The end of input is denoted by an "empty" configuration consisting solely of '.' characters; do not process this map.

```
..WO....
..WO....
BBOYGGWR
BBOYGGWR
..YR....
..YR....

..GY....
..BY....
ROYWRRBB
GWOWRBOW
..YG....
..OG....

........
........
........
........
........
........
```

H.3 Output

For each cube, output on a single line a sequence of moves that will solve the cube. Output 'X' for a turn about the X-axis, 'Y' for a turn about the Y-axis, and 'Z' for a turn about the Z-axis. Any sequence of moves (that is reasonably finite) which solves the given configuration will do. (After all, Sonny does need to execute your commands to verify that your program works!) A blank line will suffice for an input cube that is already solved.

Note that the time limit for this problem is more generous than others, and one of the more difficult configurations is provided in the sample. An example of correct output that corresponds to the provided input would be:

```
X
YZXXXZYZXYXYZZYZZYZXYY
```