The CENTRE for EDUCATION
in MATHEMATICS and COMPUTING

2007 Canadian Computing Competition: Senior Division

Sponsor:

University of
Waterloo

Canadian Computing Competition
Student Instructions for the Senior Problems

1. You may only compete in one competition. If you wish to write the Junior paper, see the other problem set.

2. Be sure to indicate on your **Student Information Form** that you are competing in the **Senior** competition.

3. You have three (3) hours to complete this competition.

4. You should assume that

   - all input is from a file named `sX.in`, where X is the problem number ($1 \leq X \leq 5$).
   - all output is to a file named `sX.out`, where X is the problem number ($1 \leq X \leq 5$).

   None of the problems require prompting: please *do not* prompt the user. Be sure your output matches the output in terms of order, spacing, etc. IT SHOULD MATCH EXACTLY!

5. Do your own work. Cheating will be dealt with harshly.

6. Do not use any features that the judge (your teacher) will not be able to use while evaluating your programs.

7. Books and written materials are allowed. Any machine-readable materials (like other programs which you have written) are *not* allowed. However, you are allowed to use "standard" libraries for your programming languages; for example, the STL for C++, java.util.*, java.io.*, etc. for Java, and so on.

8. Applications other than editors, compilers, debuggers or other standard programming tools are **not** allowed. Any use of other applications will lead to disqualification.

9. Please use file names that are unique to each problem: for example, please use `s1.pas` or `s1.c` or `s1.java` (or some other appropriate extension) for Problem S1. This will make the evaluator's task a little easier.

10. Your program will be run against test cases other than the sample ones. Be sure you test your program on other test cases.

11. Note that the top 2 Senior competitors in each region of the country will get a plaque and $100, and the schools of these competitors will also get a plaque. The regions are:

    - West (BC to Manitoba)
    - Ontario North and East
    - Metro

- Ontario Central and West
- Quebec and Atlantic

12. If you finish in the top 20 competitors in the Senior contest, you will be invited to participate in CCC Stage 2, held at the University of Waterloo from May 28-June 1, 2007. Should you finish in the top 4 at Stage 2, you will be on the team that will represent Canada at IOI 2007, held in Croatia. Note that you will need to know C, C++ or Pascal if you are invited to Stage 2. But first, do well on this contest!

13. Check the CCC website at the end of March to see how you did on this contest, how the problems were meant to be solved, and to see who the prize winners are. The CCC website is:

`www.cemc.uwaterloo.ca/ccc`

Problem S1: Federal Voting Age

Problem Description

For the big election on February 27, 2007, the government has commissioned an electronic voting system, and you have been hired as a sub-subcontractor for this very grand programming project.

Your task is to write the system that determines whether a given person is old enough to vote. The voting age is 18, so given someone's birthday, you must determine whether that person will be 18 years of age on the day of the election.

Input Specification

The input will consist of a number n ($1 \leq n \leq 100$) on a single line, indicating the number of birthdays to evaluate. Then, each of the following n lines, will be of the form y m d, where y is the year of a potential voter's birth ($0 \leq y \leq 2007$), m ($1 \leq m \leq 12$) is the month of birth, and d ($1 \leq m \leq 31$) is the day. It is assured that each birthday is a correct and valid date.

Output Specification

For each date in the input, output a line with either "Yes" if the voter is eligible to vote, or "No" otherwise.

Sample Input

```
5
1933 10 29
1989 2 28
1961 11 23
1999 12 31
1989 2 27
```

Output for Sample Input

```
Yes
No
Yes
No
Yes
```

Problem S2: Boxes

Problem Description

Nowadays, almost any item can be bought and sold on the internet. The problem is shipping. Before an item can be sent, it must be carefully packaged in a cardboard box to protect it.

While items come in many shapes and sizes, finding a box just the right size can be a problem. If the box is too small, the item will not fit. If the box is unnecessarily big, shipping cost will be higher, and the item is more likely to move around inside the box, and it may break.

Cardboard box manufacturers offer a fixed set of standard box sizes. Your task is to find the standard box size with the smallest volume into which an item will fit.

Each box is a rectangular prism with a given length, width, and height. Each item is also a rectangular prism with a given length, width, and height. An item may be rotated by multiples of 90 degrees in any direction before being packed into a box, but when it is packed, its faces must be parallel to the faces of the box. An item will fit into a box as long as its dimensions are equal to or less than the dimensions of the box.

Input Specification

The first line of input will contain a single integer $n, 0 < n < 1000$, the number of different sizes of boxes available. The next n lines will contain three integers each, giving the length, width, and height of a box. The following line will contain a single integer $m, 0 < m < 1000$, the number of items to be packaged. The next m lines will contain three integers each, giving the length, width, and height of an item. All dimensions will be in millimetres and in the range from 1 mm to 2000 mm.

Output Specification

Output is to consist of m lines, one for each item in the input. For each item, output a line containing a single integer, the volume (in mm^3) of the smallest box into which the item will fit. The same size of box may be reused for any number of items. If an item does not fit in any box, print the line: `Item does not fit.`

Sample Input

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

Output for Sample Input

```
6
24
24
60
Item does not fit.
```

Problem S3: Friends

Problem Description

In a certain school, it has been decided that students are spending too much time studying and not enough time socializing. To address this situation, it has been decided to give every student a friend. Friendship is one-way. That is, if Janet is assigned to be Sarah's friend, Janet must be friendly to Sarah, but Sarah is not required to reciprocate.

The assignment of friends is done by computer using student numbers. Every student is assigned exactly one friend. Sometimes, a 'circle of friends' occurs. For example, if Marc is assigned Fred, Fred is assigned Lori, Lori is assigned Jean, and Jean assigned Marc, we have a circle of 4 friends containing Marc, Fred, Lori, and Jean. In the circle, we can say that Marc has a separation of 0 from Fred, of 1 from Lori, of 2 from Jean, and of 3 from Marc.

Your task it to identify, given the computer assignment of friends, whether two students are in the same circle of friends, and if so, determine their separation.

Input Specification

Input begins with a single integer n ($2 \leq n \leq 9999$), on a line by itself, indicating the number of students in the class. The next n lines contain the computer assignment of friendship. An assignment is of the form $x\ y$ (where $1 \leq x \leq n, 1 \leq y \leq n, x \neq y$). For example, 1234 8765 is a possible friendship assignment indicating that student 1234 must be friends with student 8765.

Following the friendship assignments, there are a series of lines containing two student numbers, separated by a single whitespace. These lines represent the pairs of students that you will determine if they are in same circle of friends and, if so, their separation. The last line of input can be identified by the use of the 0 0 as the friend assignment.

Output Specification

For each case, you are to print, on a separate line, either `Yes` or `No` depending on whether they are in the same circle of friends. If the answer is `Yes`, follow the output `Yes` with a single whitespace and then an integer indicating the friend's separation.

Sample Input

```
6
1 2
2 3
3 1
10 11
100 10
11 100
1 100
2 3
0 0
```

Output for Sample Input

```
No
Yes 0
```

Problem S4: Waterpark

Problem Description

The local waterpark has a great slide complex, with many paths crisscrossing down the hill. There is one start point and one end point, but at various points one can turn and take different paths. Walter and Wanda are wondering exactly how many different ways there are to go down the slide. Can you solve their problem?

More precisely, there are n marked points (including the start at 1 and the end at n) where the paths down the hill may split or merge. All paths move down the hill to higher numbered positions; some paths will actually cross over others without meeting but we don't have to worry about that. We won't worry about the collisions between sliders that can happen either. Our problem is simply to determine the number of different sequences of marked points we can follow down the hill.

For example, at one small waterpark, there are 4 points with direct slides from 1 to points 2 and 4; from 2 to 3 and 4; and from 3 to 4. There are 3 ways down the hill. You can check this by seeing that we can go (1,2,3,4), (1,2,4) or (1,4).

(Here is a hint: think about starting at the bottom of the slide.)

Input Specification

Input begins with a single integer n ($1 \leq n \leq 9999$), on a line by itself, indicating the number of marked points. The next n lines contain point pairs of the of the form x y, where $1 \leq x < y \leq n$. For example, 1234 8765 indicates a direct slide from point 1234 to point 8765. The last line of input will be indicated by point pair 0 0.

Output Specification

The output is an integer, which is the number of different paths from point 1 to point n. You can assume that the number of paths will be less than 2^{30}. It is possible that there is no path from point 1 to point n, in which case the number of paths is 0.

Sample Input

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

Output for Sample Input

```
3
```

Problem S5: Bowling for Numbers

Problem Description

At the Canadian Carnival Competition (CCC), a popular game is *Bowling for Numbers*. A large number of bowling pins are lined up in a row. Each bowling pin has a number printed on it, which is the score obtained from knocking over that pin. The player is given a number of bowling balls; each bowling ball is wide enough to knock over a few consecutive and adjacent pins.

For example, one possible sequence of pins is: `2 8 5 1 9 6 9 3 2`

If the Alice was given two balls, each able to knock over three adjacent pins, the maximum score Alice could achieve would be 39, the sum of two throws: $2 + 8 + 5 = 15$, and $9 + 6 + 9 = 24$.

Bob has a strategy where he picks the shot that gives him the most score, then repeatedly picks the shot that gives him the most score from the remaining pins. This strategy doesn't always yield the maximum score, but it is close. On the test data, such a strategy would get a score of 20%.

Input Specification

Input consists of a series of test cases. The first line of input is t, $1 \leq t \leq 10$, indicating the number of test cases in the file. The first line of each test case contains three integers n k w. First is the integer n, $1 \leq n \leq 30000$, indicating the number of bowling pins. The second integer, k, $1 \leq k \leq 500$, giving the number of bowling balls available to each player. The third and final integer is w, $1 \leq w \leq n$, the width of the bowling ball (the number of adjacent pins it can knock over). The next n lines of each test case each contain a single non-negative integer less than 10000, giving the score of the pins, in order. 20% of the test data will have size $n \leq 50$.

Output Specification

For each test case, output the maximum achievable score by the player. This score is guaranteed to be less than one billion.

Sample Input

```
1
9 2 3
2
8
5
1
9
6
9
3
2
```

Output for Sample Input

```
39
```