Programming Contest, LOOK UP - EAFIT

October 3, 2015

General Comments

- This set has 7 problems.
- The contest has a total duration of 180 minutes.
- The scoreboard will be frozen after 150 minutes.
- Submissions answers won't be known if they were made after 170 minutes.
- If you are coding in Java, remember that the source file and the class containing the *main* method should be named as described in the problem statement.
- If your are using Java, **DO NOT** include any package in your source file.
- All problems were taken from several *Red de Programación Competitiva* contests.
- Good luck and enjoy!

Page intentionally left (almost) in blank.

Problem A. Dice Game

Input:	`standard input`
Output:	`standard output`
Source file name:	`dice.java`, `dice.cpp`, `dice.c`

Gunnar and Emma play a lot of board games at home, so they own many dice that are not normal 6-sided dice. For example they own a die that has 10 sides with numbers 47, 48, ..., 56 on it.

There has been a big storm in Stockholm, so Gunnar and Emma have been stuck at home without electricity for a couple of hours. They have finished playing all the games they have, so they came up with a new one. Each player has 2 dice which he or she rolls. The player with a bigger sum wins. If both sums are the same, the game ends in a tie.

Your task is: Given the description of Gunnar's and Emma's dice, which player has higher chances of winning?

All of their dice have the following property: each die contains numbers $a, a+1, ..., b$, where a and b are the lowest and highest numbers respectively on the die.

Each number appears exactly on one side, so the die has $b - a + 1$ sides.

Input

The first line contains four integers a_1, b_1, a_2, b_2 that describe Gunnar's dice. Dice number i contains numbers $a_i, a_i + 1, ..., b_i$ on its sides. You may assume that $1 \leq a_i \leq b_i \leq 100$. You can further assume that each die has at least four sides, so $a_i + 3 \leq b_i$.

The second line contains the description of Emma's dice in the same format.

Output

Output the name of the player that has higher probability of winning. Output "Tie" if both players have same probability of winning.

Sample input and output

standard input	standard output
1 4 1 4 1 6 1 6	Emma
1 8 1 8 1 10 2 5	Tie
2 5 2 7 1 5 2 5	Gunnar

Problem B. Upside down primes

Input:	`standard input`
Output:	`standard output`
Source file name:	`primes.java`, `primes.cpp`, `primes.c`

Last night, I must have dropped my alarm clock. When the alarm went off in the morning, it showed 51:80 instead of 08:15. This made me realize that if you rotate a seven segment display like it is used in digital clocks by 180 degrees, some numbers still are numbers after turning them upside down.

Figure B-1: Prime number 18115211 on a seven segment display (see third sample).

Figure B-2: 18115211 turned upside down (i.e. rotated by 180 degrees) gives 11251181, whichis not prime.

As you can see,

- 0, 2, 5, and 8 still are 0, 2, 5, and 8.
- 1 is still readable as 1 (only moved left).
- 6 turns into 9, while 9 turns into 6.
- 3, 4 and 7 are no longer valid numbers (E, h and L)

My favourite numbers are primes, of course. Your job is to check whether a number **is a prime** and **still a prime** when turned upside down.

Input

One line with the integer N in question ($1 \leq N \leq 10^{16}$). N will not have leading zeros.

Output

Print one line of output containing "yes" if the number **is a prime** and **still a prime** if turned upside down, "no" otherwise.

Sample input and output

standard input	standard output
151	yes
23	no
18115211	no

Problem C. Tri-du

Input:	`standard input`
Output:	`standard output`
Source file name:	`tridu.java`, `tridu.cpp`, `tridu.c`

Tri-du is a card game inspired in the popular game of Truco. The game uses a normal deck of 52 cards, with 13 cards of each suit, but suits are ignored. What is used is the value of the cards, considered as integers between 1 to 13.
In the game, each player gets three cards. The rules are simple:

- A Three of a Kind (three cards of the same value) wins over a Pair (two cards of the same value).
- A Three of a Kind formed by cards of a larger value wins over a Three of a Kind formed by cards of a smaller value.
- A Pair formed by cards of a larger value wins over a Pair formed by cards of a smaller value.

Note that the game may not have a winner in many situations; in those cases, the cards are returned to the deck, which is re-shuffled and a new game starts.
A player received already two of the three cards, and knows their values. Your task is to write a program to determine the value of the third card that maximizes the probability of that player winning the game.

Input

The input contains several test cases. In each test case, the input consists of a single line, which contains two integers A ($1 \leq A \leq 13$) and B ($1 \leq B \leq 13$) that indicates the value of the two first received cards.

Output

For each test case in the input, your program must produce a single line, containing exactly one integer, representing the value of the card that maximizes the probability of the player winning the game.

Sample input and output

standard input	standard output
10 7 2 2	10 2

Problem D. Extreme Sort

Input:	`standard input`
Output:	`standard output`
Source file name:	`sort.java`, `sort.cpp`, `sort.c`

John likes sorting algorithms very much. He has studied quicksort, merge sort, radix sort, and many more.
A long time ago he has written a lock-free parallel string sorting program. It was a combination of burstsort and multi-key quicksort. To implement burstsort you need to build a tree of buckets.
For each input string you walk through the tree and insert part of the string into the right bucket. When a bucket fills up, it "bursts" and becomes a new subtree (with new buckets).

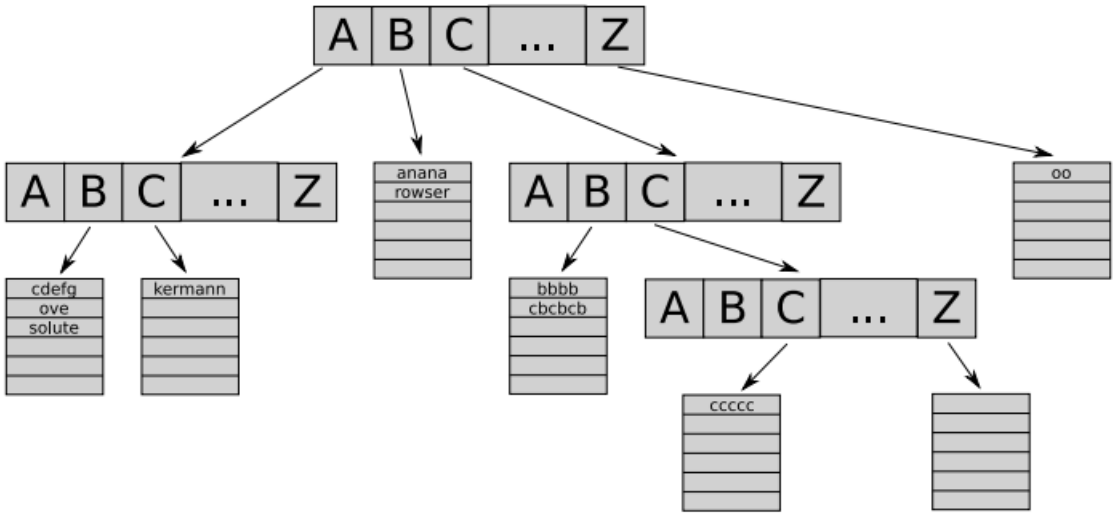

Figure D-1: Burstsort data structure.

Well, enough about the past. Today John is playing with sorting algorithms again. This time it's numbers. He has an idea for a new algorithm, "extreme sort". It's extremely fast, performance levels are OVER NINETHOUSAND. Before he tells anyone any details, he wants to make sure that it works correctly.
Your task is to help him and verify that the so-called extreme property holds after the first phase of the algorithm. The extreme property is defined as $min(x_{i,j}) \geq 0$, where

$$x_{i,j} = \begin{cases} a_j - a_i & \text{for} \quad 1 \leq i < j \leq N \\ 9001 & \text{otherwise} \end{cases}$$

Input

The first line contains a single integer N ($1 \leq N \leq 1024$). The second line contains N integers $a_1\ a_2\ \ldots\ a_N$ ($1 \leq a_i \leq 1024$).

Output

Print one line of output containing "yes" if the extreme property holds for the given input, "no" otherwise.

Sample input and output

standard input	standard output
2 1 2	yes
4 2 1 3 4	no

Problem E. Bus

Input:	`standard input`
Output:	`standard output`
Source file name:	`bus.java`, `bus.cpp`, `bus.c`

A bus with n passengers opens its door at the bus stop. Exactly half of its passengers and an additional half of a passenger get out. On the next stop, again, half of the passengers plus half of a passenger leave the bus. This goes on for k stops in total. Knowing that the bus leaves the last stop empty, and that no one was hurt during the trip, determine the initial number n of people in the bus.

Input

The first line of input contains the number of test cases T. The descriptions of the test cases follow: The only line of each test case contains the number of stops $k, 1 \leq k \leq 30$.

Output

For each test case, output a single line containing a single integer - the initial number of bus passengers.

Sample input and output

standard input	standard output
2 1 3	1 7

Problem F. Factorial

Input:	`standard input`
Output:	`standard output`
Source file name:	`factorial.java`, `factorial.cpp`, `factorial.c`

The *factorial* of a positive integer number N, denoted as $N!$, is defined as the product of all positive integer numbers smaller or equal to N. For example $4! = 4 \times 3 \times 2 \times 1 = 24$.
Given a positive integer number N, you have to write a program to determine the smallest number k so that $N = a_1! + a_2! + \ldots + a_k!$, where every a_i, for $1 \leq i \leq k$, is a positive integer number.

Input

The input consists of several test cases. A test case is composed of a single line, containing one integer number N ($1 \leq N \leq 105$).

Output

For each test case in the output your program must output the smallest quantity of factorial numbers whose sum is equal to N.

Sample input and output

standard input	standard output
10 25	3 2

Problem G. What does the fox say?

Input:	`standard input`
Output:	`standard output`
Source file name:	`fox.java`, `fox.cpp`, `fox.c`

Determined to discover the ancient mystery - the sound that the fox makes - you went into the forest, armed with a very good digital audio recorder. The forest is, however, full of animals- voices, and on your recording, many different sounds can be heard. But you are well prepared for your task: you know exactly all the sounds which other animals make. Therefore the rest of the recording - all the unidentified noises - must have been made by the fox.

Input

The first line of input contains the number of test cases T. The descriptions of the test cases follow: The first line of each test case contains the recording - words over lower case English alphabet, separated by spaces. Each contains at most 100 letters and there are no more than 100 words. The next few lines are your pre-gathered information about other animals, in the format `<animal> goes <sound>`. There are no more than 100 animals, their names are not longer than 100 letters each and are actual names of animals in English. There is no `fox goes ...` among these lines.

The last line of the test case is exactly the question you are supposed to answer: `what does the fox say?`

Output

For each test case, output one line containing the sounds made by the fox, in the order from the recording. You may assume that the fox was not silent (contrary to popular belief, foxes do not communicate by Morse code).

Sample input and output

standard input	standard output
1 toot woof wa ow ow ow pa blub blub pa toot pa blub pa pa ow pow toot dog goes woof fish goes blub elephant goes toot seal goes ow what does the fox say?	wa pa pa pa pa pa pow