University Interscholastic League

Computer Science Competition

2014 State Programming Problem Set

DO NOT OPEN THIS PACKET UNTIL INSTRUCTED TO BEGIN!

I. General Notes

1. Do the problems in any order you like. They do not have to be done in order from 1 to 12.

2. All problems have a value of 60 points. Incorrect submissions may be reworked and resubmitted, but will receive a deduction of 5 points for each incorrect submission. Deductions are only included in the team score for problems that are ultimately solved correctly.

3. There is no extraneous input. All input is exactly as specified in the problem. Unless specified by the problem, integer inputs will not have leading zeros. Unless otherwise specified, your program should read to the end of file.

4. Your program should not print extraneous output. Follow the form exactly as given in the problem.

II. Table of Contents

1. GCD

Program Name: GCD.java Input File: gcd.dat

The greatest common divisor of two integers is the largest integer that evenly divides both of them. For example, gcd(6, 10) is 2. More generally, the greatest common divisor of N integers is the largest integer that divides all of them evenly. For example, gcd(30, 45, 105) = 15. Given a list of numbers, your job is to write a program that finds the largest integer that divides every integer in the list.

Input
The first line will contain the number of test cases T.
The first line of each test case contains a single integer N, the number of numbers in this test case.
The second line of each test case contains N integers separated by spaces on a single line.

Output
For each test case, output the largest integer that divides all of the numbers given.

Constraints
```
1 <= T <= 10
2 <= N <= 20
1 <= each number <= 1,000,000
```

Example Input File
```
3
2
6 10
3
45 30 105
4
2 3 4 5
```

Example Output to Screen
```
2
15
1
```

2. Goodbye

Program Name: Goodbye.java Input File: goodbye.dat

As this is the last contest of UIL 2014, we want to bid you goodbye. So, given a list of strings tell us how many "goodbye"'s are present in each string.
Note: "goodbye"'s are case insensitive.

Input
The first line will contain the number of test cases T.
Each test case contains one string, on its own line.

Output
Output a single integer containing the number of "goodbye"'s in the string.

Constraints
```
1 <= T <= 10
1 <= Length(String) <= 1000
```

Example Input File
```
2
Googooogoodbyegoogoo
GoodbyegoodByegooby
```

Example Output to Screen
```
1
2
```

Explanation of the example
In the first case, only one goodbye is found: Googooo(goodbye)googoo, so the answer is 1.
In the second case, there are two different goodbyes: (Goodbye)goodByegooby and Goodbye(goodBye)gooby, so the answer is 2.

3. Inside

Program Name: Inside.java Input File: inside.dat

Your friend Edith has recently been taking part in geocaching, and has realized that some of her goals are actually on her property. The problem is that her property is irregularly shaped, so she needs your help figuring out if each location is on her property. You will be given a list of points in form "X Y", giving x and y coordinates that form the corners of a polygon. Each point will be connected to the points given immediately before and after it, with the first and last points also being connected. You will then be given a list of points, which for each one you will print "inside" or "outside" depending on if that point is inside or outside of the polygon.

Input
The first line will contain a single integer T, stating the number of test cases to follow. Each test case will start with two integers C and P that refer to the number of corners of the polygon and the number of queried points respectively. C lines will follow, each with two integers X and Y, and will form the vertices of the polygon. P lines will follow after this, each stating a point in the form X and Y (integers) that you must determine if it is inside or outside of the specified polygon. There will be no point on the boundary of the polygon.

Constraints
```
0 < T < 5
2 < C <= 10
0 < P <= 10
-100 < X < 100
-100 < Y < 100
```

Output
Your output should contain a list of answers each on its own line. Each answer is either "inside" or "outside" depending on whether the point is inside or outside of its corresponding polygon.

Example Input File
```
2
3 2
-1 -1
0 1
1 -1
0 0
1 1
4 3
-1 -1
-1 1
1 1
1 -1
0 0
1 2
-3 -3
```

Example Output to Screen
```
inside
outside
inside
outside
outside
```

4. Knight

Program Name: Knight.java Input File: knight.dat

In chess the knight piece can only move in a special fashion. This movement involves moving two places up, down, left, or right, and then one place perpendicular to its previous direction. Only one piece can occupy a chessboard square -- if a knight lands on a piece occupied by an enemy piece, it captures the enemy piece, which is then removed from the board.

You will be given a list of locations of pieces on an NxN board. Your knight will start at the bottom left location (0,0) (there will be no other piece at (0,0)) and must always move one or two steps towards the right in each move. You must solve for the most pieces that your knight can capture by the time it reaches the right side of the board if it is the only piece moving and all other pieces are enemies.

Input
The first line will contain two numbers N, and P. P lines will follow, each stating two points X and Y.

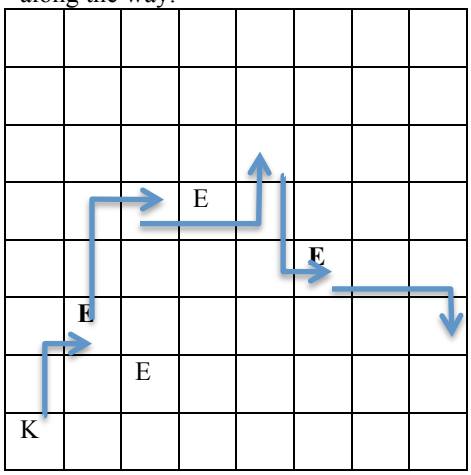
Constraints
```
4 < N < 100
0 < P < 100
0 <= X,Y < N
```

Output
Output a single number stating the maximum number of pieces you can take.

Example Input File
```
5 3
2 1
1 2
3 3
```

Example Output to Screen
```
2
```

This figure shows one path to that the knight takes to reach the right side of the board, capturing 2 pieces (in Bold) along the way.



5. Last Level

Program Name: LastLevel.java Input File: lastlevel.dat

For your upcoming family reunion, you decided to make a family tree diagram, starting from one of your ancestors. Spouses and partners are not included; only direct descendants. You were given a list of all the ancestor's N-1 descendants, with a number assigned to each person. You are also given a list of pairs of people "x y" who have a parent-child relationship between them, with x being the parent.

As you start drawing up the tree, you wonder who in your family has no children. Can you write a program to help you find that out?

Input

The first line will contain the number of test cases T.
The first line of each test case contains the single integer N, the number of people, including your first ancestor.
The next N-1 lines of each test case contain a parent-child relationship in the form “x y”.

Output

Output a single line of integers separated by spaces, which are all of the childless people. They should be output in increasing order. Print an extra space after the last leaf.

Constraints

$1 <= T <= 10$
$1 <= N <= 10$

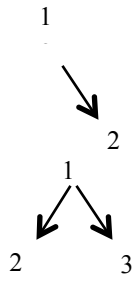
Example Input File

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

Example Output to Screen

```
2
2 3
4
```

Explanation of the example:
For the first case, we have a family of 2 people, with 1 being 2’s parent:
Thus, 1 is the parent of 2, but 2 doesn’t have a child.
For the second case, we have a family of 3 people, where 1 is the parent of 2 and 3:



6. Optimize

Program Name: Optimize.java Input File: optimize.dat

You have been hired by Home Depot, and they want help deciding the optimal pipe lengths to sell. Long pipes come in to the store from the wholesaler, and are cut to different lengths, before the pieces are sold. Different pipe lengths are worth different amounts of profit. Your job is to decide how to maximize profit by deciding how to cut up the long pipe.

Input
The first line will contain T, the number of test cases to follow. T pairs of lines will follow. The first line will contain a list of space separated numbers. The first number is the profit obtained from a pipe of length 1 foot, the second from a length of 2 feet, etc. The next line will contain a single integer L, stating the length of pipe in feet that you will start with.

Constraints
$0 < T < 20$
$0 < L < 300$

Output
Your output will be a single integer. The integer will be the maximum profit that can be obtained from the given pipe.

Example Input File
```
2
1 4 2 3
4
1 2 3
8
```

Example Output to Screen
```
8
8
```

Explanation
For the first testcase, the most profit is made by dividing the 4-foot pipe into 2 pieces, each 2 feet long.
For the second case, the 8-foot pipe should be cut into 3 pieces, of 2-foot, 3-foot and 3-foot respectively.

7. Ping Pong

Program Name: PingPong.java Input File: pingpong.dat

Imagine you are working for a professional ping pong tournament. To add a little more fun to the tournament, management has asked you to make statistical predictions about who will win each match, so they can make the odds for their bracket pick-em.

They have given you a bunch of rallies to analyze. As you know, in ping pong, two players hit a ball to each other across a net. As long as the ball is in play, it is called a rally. When a player misses the ball or hits it incorrectly, he loses the point, and that also then ends the rally.

Each match consists of two players. Player 1 is the player that serves, and Player 2 is the player that receives the serve. Management has estimated the "skill" of each player as the probability that when they hit the ball, they score. Management wants you to calculate the probability that player 1 wins the rally. This is calculated by the probability that player one wins the rally on his first hit plus on his second hit + …. Given that this is an infinite series, and it takes a while to add up infinite numbers of elements, you just decide to try to find the probability that player 1 wins on his nth hit (and not before). For example: if player one has a skill of 0.6 and player 2 has a skill of 0.3, the probability that player one wins the game on his first hit is .6. The probability that he wins the rally on his second hit takes into account the probability that he scores on the his second hit, the probability that he didn't score on his first hit (because the rally would have ended on the first hit), and likewise the probability that player 2 didn't score on his first hit as well. Thus, the probability that player 1 wins on his second hit is (1-0.6)*(1-0.3)*0.6 = 0.168. And so on.

Input

The first line will contain the number of test cases T.
Each test case contains one line, in the form of p1 p2 n. p1 is a floating point number that represents the skill of p1, p2 is the skill of p2, and n is an integer telling you which hit you need to calculate the probability of p1 winning.

Output

Output a single floating point number, which is the probability that player one wins on his nth hit. You should output the answer rounded to 4 decimal places. You will be allowed a tolerance of ± 0.0001 on each answer.

Constraints

```
1 <= T <= 10
0.0 <= p1 <= 1.0
0.0 <= p2 <= 1.0
0 <= h <= 10
```

Example Input File

```
3
0.6 0.3 1
0.6 0.3 2
1.0 0.7 2
```

Example Output to Screen

```
0.6000
0.1680
0.0000
```

Explanation of the example

The first two are explained in the description. In the last case, the probability that player 1 wins when he hits the ball is 1. Thus, he is always going to win on his first hit, so he can never win on his second hit.

8. Rearrange

Program Name: Rearrange.java Input File: rearrange.dat

Reverse Polish notation (RPN) calculators are considered to be optimal calculators to use because they never require parentheses. This is done by using a stack-based approach, where each operator occurs after all of its operands. You will be converting from the standard infix form of equations to this described postfix form. This involves rearranging the operators and operands so that they evaluate correctly without thinking of operator precedence. Your algorithm will only have to work with + - * /.

An advantage of RPN is that it obviates the need for parentheses that are required by infix. While "3 - 4 * 5" can also be written "3 - (4 * 5)", that means something quite different from "(3 - 4) * 5". In postfix, the former could be written "3 4 5 * -", which unambiguously means "3 (4 5 *) -" which reduces to "3 20 -"; the latter could be written "3 4 - 5 *" (or 5 3 4 - *, if keeping similar formatting), which unambiguously means "(3 4 -) 5 *".

One example of the desired conversion is “3 + 4” :: “3 4 +”. Another, more complicated example is “3 + 2 * 5” :: “3 2 5 * +”.

Input
The first line contains a single integer, T, which states the number of test cases to follow. Each test case will consist of a single line that contains a function in infix form, using only integers. There will be no parentheses, and the usual Java operator precedence rules apply.

Constraints
`0 < T < 30`
The number of operands `< 100`
The number of operators `< 100`
`0 <` each number `< 1000`

Output
For each test case, print out on its own line the function in postfix form. All numbers must be given in the same order as given, but the order of the operators may vary from their order in the infix form. There is a unique solution for each problem. You should output a single space between each number/operator.

Example Input File
```
2
4 * 2 + 1
3 + 1 / 7
```

Example Output to Screen
```
4 2 * 1 +
3 1 7 / +
```

9. Switch

Program Name: Switch.java Input File: switch.dat

Your boss recently purchased a bunch of chandeliers from a strange company. These chandeliers consist of a single row of n light bulbs. The annoying thing about these chandeliers is that there isn't a single switch that turns all of the bulbs on or off. Instead, they come with k toggle switches, and flipping a switch flips an arbitrary set of bulbs from OFF to ON or ON to OFF i.e. it toggles the lights. Fortunately, it is easy to determine which bulbs each switch flips by turning them on and off one at a time. So, you wrote down what each switch does in the format of a string n long, with each position consisting of a Y if that switch flips that light, and an N if that switch doesn't affect the state of that light. Thus if switch k operates bulb n, it means if when k is flipped, bulb n goes OFF if it was already ON, and goes ON if it was OFF.

Your boss wants to know the minimum number of switch flips it takes to turn all of the lights in each chandelier on, from a starting position of all OFF. Or, if it takes more than 10 switch flips, the boss deems the chandelier "lame" and "too hard to use", so you throw it out. Write him a program that tells him how many flips it takes, or if it is too many or impossible to do, output "TOO HARD!".

Input
The first line of input contains the number of chandeliers to process.
The first line of each chandelier contains two integers: n and k, for $0 < n \leq 100$, $0 < k \leq 20$.
The next k lines each consist of a single string, n characters long, composed of only Y's and N's as described above.

Output
For **each** chandelier, if the minimum number of switch flips required is ≤ 10, output that number; otherwise output the string "TOO HARD!"

Example Input File

```
3
4 3
YNNN
NYNN
NNYY
4 4
YYNN
YNNY
NNYN
YNYN
11 11
YNNNNNNNNNN
NYNNNNNNNNN
NNYNNNNNNNN
NNNYNNNNNNN
NNNNYNNNNNN
NNNNNYNNNNN
NNNNNNYNNNN
NNNNNNNYNNN
NNNNNNNNYNN
NNNNNNNNNYN
NNNNNNNNNNY
```

Example Output to Screen

```
3
3
TOO HARD!
```

Explanation of sample case:
We will represent the string of lights as a 0 or 1 for each light: 0 if the light is off and 1 if it is on.

First case: Start with 0000
Flip switch 1 (YNNN), get 1000
Flip switch 2 (NYNN), get 1100
Flip switch 3 (NNYY), get 1111
All lights are on, only took 3 flips.

Second case: Start with 0000
Flip switch 1 (YYNN), get 1100
Flip switch 2 (YNNY), get 0101
Flip switch 4 (YNYN), get 1111
All lights are on, took 3 flips

Third case: start with 00000000000
Each switch turns exactly one light on, and we need 11 lights on, so there are no 10 flips you can choose that result in all of the lights on, so you output "TOO HARD!"

10. VC1

Program Name: VC1.java Input File: vc1.dat

In distributed computing, there is a concept known as a "vector clock". Assume you have a distributed system consisting of k processes, and each process runs a series of events, and communicates with the other processes by sending messages. These events can either be local events, a send event, or a receive event. The vector clock of an event e contains, for each process, the entire history of events that "happen before" e. Instead of a long list of events though, you can show that there is one last event on each process that happens before e, so you can represent the entire history by storing the event number of the last event of each process that happens before e in a vector, creating a "vector clock". With this, we can build a relation between vector clocks called "happens-before", or à. The syntax we will use is: given p processes, a vector clock VC for an event e is an array of integers of length p, where VC[i] is the last event number on process i that **"happened-before"** e. For example, for event e = event 3 on process 1 (0-indexed), with 3 total processes, the vector clock could be [2, 3, 1], or [0, 3, 0], or [4, 1, 7], etc….
Vector clocks have the following properties with respect to the "happens-before" relation:
If two events are on the same process, then the earlier event à the later event.
If event a is the send event for a message M and event b is the receive event for a message M, then a à b.
If a à b and b à c, then a à c (transitivity).
Given two events a and b, if neither a à b nor b à a, then they are said to be "concurrent", or a || b.
.
Since vector clocks contain the last event that "happens-before" e on each process, given two vector clocks A and B for events a and b respectively, a à b when for all processes i, A[i] ≤ B[i]. In other words, for every entry i in the vector clock, the last event number A has for process i must be less than or equal to the last event number B has for process i.

Also, if no event on process i has à e, we say E[i] = 0.
Given the vector clocks of two events a and b, your job is to figure out whether a à b, b à a, or a || b (they are 'concurent' -- neither a à b nor b à a) .

Input
The first line will contain the number of test cases T.
Each test case contains three lines. The first line is the number of processes P. The second line is the vector clock A represented by P space-separated integers. The third line is the vector clock B, in the same format as A.

Output
Output "`A -> B`" if A à B, "`B -> A`" if B à A, and "`A || B`" if they are concurrent. It is guaranteed that A and B will never be equal, so exactly one of these is always true.

Constraints
```
1 <= T <= 10
0 <= P <= 20
1 <= each event number <= 10^9
```

Example Input File

```
3
2
1 1
2 1
3
3 3 3
1 2 3
2
1 2
2 1
```

Example Output to Screen

```
A -> B
B -> A
A || B
```

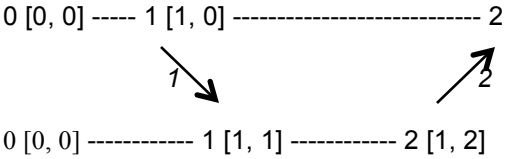
Explanation of the example

In the first test case, all numbers in A are ≤ all numbers in B. In the second case, all numbers in B are ≤ all numbers in A. In the third case, A[0] < B[0], but A[1] > B[1], so they are concurrent.

11. VC2

Program Name: VC2.java Input File: vc2.dat

Building off of the explanation of vector clocks in the previous problem, we can see how the vector clocks of events are updated in a distributed system as it completes its computation.
In this problem, we will represent the vector clocks as a sequence of numbers, separated by commas, in square brackets.
Say there is a new event on a process. If the event is not a receive event, then the only change that needs to be made is incrementing the event number for the current process it happens on. For example, if a non-receive event happens on the first process (0-indexed), and the last event that happened on that process had a vector clock of [2, 3, 4], this new event would have a vector clock of [3, 3, 4].
If the event is a receive, we have a slightly more complicated process. Let r be our receive event, s be the corresponding send event, and l be the previous event on this process. For each process i, r[i] = max(s[i], l[i]), and afterwards, you must increment the event number of the local counter. For example, look at the following process execution of two processes communicating with a pair of messages:



In the above figure, process 0 sends a message *1* to process 1, and process 1 replies with *2*. The format is x VC, where x is the event number of that event, and VC is the vector clock of the event. We also give each message a message id. We assume each process has a “start” event of 0.

To look at an example, let’s figure out the vector clock of event 2 on process 0. In this case, event 2 on process 0 is a receive event, and the corresponding send event, or “s”, is event 2 on process 1.Also, event 1 on process 0l is the previous event on process 0, or “l”. Thus, the vector clock of l is [1, 0], and the vector clock of s is [1, 2], and r = [max(s[0], l[0]), max(s[1], l[1])] = [max(1, 1), max(0, 2)] = [1, 2]. After, we must increment the local counter. Since the current event is happening on process 0, we must increment r[0] by 1, from 1 to 2, so the final result is [2, 2].

Input

The first line will contain the number of test cases, T.
Each test case contains P + N + 2 lines. The first line is the number of processes P. The next P lines each contain every event in the event history of that process. Each process implicitly has a starting event 0, and every event listed will be either a send or receive. Thus each message will be SN or RN, where N is the unique message id of that message. The next line contains the number N, the number of vector clocks you are to compute. Each line after that will contain two integers p and k, meaning you need to find the vector clock of the kth event on the pth process.

Output

Each vector clock is enclosed by brackets, has the numbers separated by commas, with no spaces.

Constraints

$1 \leq T \leq 5$
$0 \leq P \leq 10$
$1 \leq N \leq 50$

Example Input File

```
2
2
S1 R2
R1 S2
1
0 2
2
S1 S3 R2 R4 S5
R1 S2 S4 R3 R5
2
1 1
0 5
```

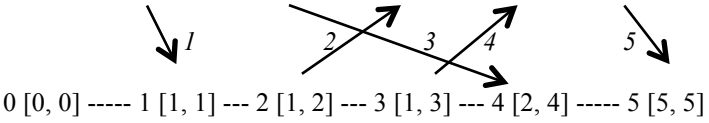
Example Output to Screen

```
[2,2]
[1,1]
[5,3]
```

Explanation of Sample Output:

The first case is the example described above.

The second and third cases are illustrated below:



The mathematical explanation of the answers is as follows:

Event 1 on process 1 is a receive event, so we need to consider the previous local event (event 0 on process 1) and the corresponding send event, event 1 on process 0. Event 0 on process 1 has a vector clock of [0, 0], and event 1 on process 0 has a vector clock of [1, 0], so we apply the receive update rule, making the new vector clock, [max(1, 0), max(0, 0)] = [1,0], and then incrementing process 1's position, giving us [1,1]

Event 5 on process 0 is a send event, so we only need to get the vector clock of the previous local event, event 4 on process 0, which has a vector clock of [4, 3], and we just need to increment position 0, resulting in [5,3].

We can also sanity check these results with the definition provided in the VC1 outline of vector clocks, or the "last event on each process that happens-before this event".

For the first case, we are looking at event 2 on process 0. At this point, the vector clock at position 0 is obviously 2, since process 0 saw its own last event. Additionally, event 2 on process 1 was the sending of message *2*, corresponding to the receive in event 2 on process 0, so that event is the latest event on process 1 to happen-before event 2 on process 0. Thus, our vector clock is [2, 2].

For the second case, event 1 on process 1, the last event it saw on process 0 was the send of message *1*, event 1 on process 0, and the last event it saw on itself was this event, so the vector clock is [1,1].

For the third case, event 5 on process 0, the last event on process 0 that process 0 has seen is 5, and the last event process 0 saw on process 1 was the send of message *4*, which was event 3 on process 1, so the vector clock is [5, 3].

12. Wordfinder

Program Name: Wordfinder.java Input File: wordfinder.dat

You are playing a game of Scrabble with one of your friends. You can't decide what word to play with the letters in your rack, so you decide to pull up an online dictionary, and see which words you can play with the letters in your rack (ignoring what is on the board). This dictionary doesn't contain many words, but it would still be useful for you to write a program to see how many words in the dictionary you can make using the letters in your rack.

Input
The first line will contain two integers: N and R, where N is the number of words in the dictionary, and R is the number of racks you will check against that dictionary.
The following N lines will contain one word per line in the dictionary.
The R lines after that will contain one string per line that contains all of the letters on your rack.

Output
For each rack, output the number of words in the dictionary you could play with the letters in your rack.

Constraints
`1 <= N <= 40`
`1 <= R <= 10`
`1` <= length of each word <= `10`
`1` <= length of rack <= `10`

Example Input File
```
4 2
god
good
bad
sad
odag
osgdao
```

Example Output to Screen
```
1
3
```

Explanation of the example
Your dictionary contains 4 words: god, good, bad, and sad.
In the first case your letters are "o,d,a,g", so you can make god, not good, not bad, and not sad, for a total of 1.
In the second case your letters are "o,s,g,d,a,o", so you can make god, good, not bad, and sad, for a total of 3.