2014 University of Florida ACM

High School Programming Contest

February 25, 2014

Problems

A. Caesar Cipher
B. Calling Drivers
C. Chair Scavenging
D. GCD
E. Multiple Networks
F. Party Poopers
G. PrimeDigit
H. Magic Van
I. Square Darts
J. Work Splitter
K. Weight Balancing

Problem A

Caesar Cipher

In this problem, you will be given a string consisting of lowercase alphabetic characters as well as digits 0-9. Your goal is to encrypt the given sequence using a Caesar cipher with the provided shift value. You will need to shift alphabetic characters and numeric characters separately.

Here are the two alphabets we will be using:

abcdefghijklmnopqrstuvwxyz
0123456789

For example, given a shift value of 2, the letter 'a' shifts to 'c', the letter 'z' shifts to 'b', the number '0' shifts to '2', and the number '9' shifts to '1'. For a shift value of -1, the letter 'a' shifts to 'z', 'z' to 'y', and '0' to '9'.

INPUT:
The input will begin with a single line containing a number T, the number of test cases to follow. There will be a large number of test cases. The next T lines will have an integer N ($-1000 \leq N \leq 1000$), and a string S ($1 \leq \textit{length of } S \leq 100$), separated by a single space. N is the shift amount desired - a positive value means shift right, and a negative value means shift left. The string S will consist of only characters *a-z* and *0-9*; no uppercase letters.

OUTPUT:
The output will consist of T lines, one for each of the given test cases. Output the result of applying the Caesar cipher for each of the given test cases.

SAMPLE INPUT:	**SAMPLE OUTPUT:**
5 1 az91 -1 az91 260 az91 1 gzudetm 52 good8luck8on8the8contest	ba02 zy80 az91 havefun good0luck0on0the0contest

Problem B

Calling Drivers

On the way back from the ACM ICPC Southeast Regional contest, the team's van broke down (true story). Here's our situation: there are N people stuck on the side of the road, and there are M drivers who can come to their rescue. The N people are all stuck at the same location. Each driver can pick up a limited number of people to drive home. Additionally, the drivers are coming from different locations, so the time to reach the stranded will be different. *Note:* all the drivers are called at the same time.

Your job is to find the minimum time it takes for all the stranded people to be picked up. There will always be enough cars to take everyone home on the first trip.

INPUT:
The file will begin with a single line containing T, the number of test cases that follow. For each test case, the first line of input will have two integers N ($1 \leq N \leq 10000$) and M ($1 \leq M \leq 1000$), separated by a single space. The next M lines will each have two integers R ($1 \leq R \leq 500$) and C ($1 \leq C \leq 100$), separated by a single space. R is the time in minutes it will take that car to reach the stranded people, and C is the number of stranded people that the car can rescue.

OUTPUT:
The output will consist of T lines, each containing the minimum time for everyone to get home for the given test case.

SAMPLE INPUT:	**SAMPLE OUTPUT:**
2 10 4 60 3 80 1 100 4 120 5 15 6 20 2 20 3 30 4 50 100 45 1 500 11	120 50

Problem C

Chair Scavenging

The UF Programming Team often runs out of chairs to sit in during practice. In order to overcome this hardship, the team borrows chairs from other adjacent rooms. Given the number of people who are coming to practice and the number of chairs in the room, how many more chairs must the team take from other rooms? Sometimes there are enough chairs to begin with. In this case, we need not take any more chairs to the room.

INPUT:

The input will begin with a single line containing T, the number of test cases to follow. On each of the next T lines there will be two integers, N ($1 \leq N \leq 10^9$) and M ($1 \leq M \leq 10^9$), separated by a single space. N is the number of people coming to practice that day and M is the number of chairs already in the room that day.

OUTPUT:

The output will consist of T lines, each containing a single number - the number of additional chairs needed for that day.

SAMPLE INPUT:	**SAMPLE OUTPUT:**
2 10 9 15 8	1 7

Problem D
GCD

The UF Programming Team wants to find the GCD of N numbers. However, we are so busy practicing that we don't have time to. Help us out by finding the GCD of N ($1 \leq N \leq 100$) numbers.

INPUT:
The first line will contain a number T, the number of test cases to follow. Each test case will begin with a number N, the amount of numbers we wish to find the GCD of. The next line will contain N integers, separated by spaces: $A_1 A_2 \ldots A_N$ ($1 \leq A_i \leq 10^9$). There may be a trailing space at the end of this line.

OUTPUT:
The output will contain T lines, one for the GCD of each of the given test cases.

SAMPLE INPUT:	**SAMPLE OUTPUT:**
3 1 4 3 3 3 3 4 4 6 8 10	4 3 2

Problem E

Multiple Networks

There are many computers at the University of Florida. Computers connect to one another to form computer networks. The UF Programming Team is trying to determine the exact amount of computer networks at the University of Florida.

Two computers A and B are in the same network if there is a sequence of direct connections starting at computer A that leads to computer B.

The team is given a list of computers and the corresponding information detailing the computers that are directly connected to the given computer. Please help the team find the number of networks.

INPUT:
The input will begin with a single line containing a number T, the number of test cases to follow. Each test case begins with a single line containing a number N ($1 \leq N \leq 10^5$), the number of computers at UF. The next N lines will contain the information about the connections, with the i-th line containing information about the connections of the i-th computer. All given connections are bidirectional, some connections may be listed multiple times, and for a connection between computers A and B, it is possible that the connection will only be listed in $B's$ list.

Each of the next N lines will begin with a number K ($0 \leq K \leq 1000$), indicating the number of computers connected to the i-th computer. K may be different for different computers. On the same line, following the K will be K numbers, signifying the ID of the computers connected to the i-th computer. In this list, numbers may be repeated, and the list may indicate that the computer connects to itself. Computer IDs range from 1 to N, and every computer has a unique ID.

OUTPUT:
The output will consist of T lines, one for each test case. Output the number of distinct networks for each test case on a separate line.

SAMPLE INPUT:	**SAMPLE OUTPUT:**
2 3 1 2 1 3 1 1 5 0 1 1 0 0 3 3 3 2	1 2

Explanation of Test Case 1:

Computer with ID1 is connected to computer with ID2 (and maybe others, depending on their lists).
Computer with ID2 is connected to computer with ID3 (and also to ID1, because of the connection above).
Computer with ID3 is connected to computer with ID1 (and also ID2, because of above).

Hence, all the computers are connected together, so there is only 1 network.

Explanation of Test Case 2:

Computer with ID1 is connected to 0 other computers (may have connections in other lists)
Computer with ID2 is connected to 1 other computer, which has ID1.
Computer with ID3 is connected to 0 other computers.
Computer with ID4 is connected to 0 other computers.
Computer with ID5 is connected to 3 other computers, which have IDs 3, 3, and 2.
ID3 was repeated, but that doesn't matter.

We have 2 networks here, ID1, ID2, ID3, and ID5 are together, and ID4 is by itself.

Problem F

Party Poopers

Every year, the UF Programming Team's faculty advisor, Dave, holds a party at his house and invites everyone on the team. Because the party is a lot of work for Dave, he only holds the party if enough members accept the invitation.

Many of the UF Programming Team members don't socialize much, and feel awkward at parties, so their natural inclination is to avoid parties like the plague. Many of the members will only go if they know some other set of people are going, whereas some members know that DaveHeim[1] is always lots of fun, and so they'll attend no matter who else is going. Other members just totally despise parties or are busy, and so they won't be going at all.

Dave is trying to figure out if enough people are planning on attending his party, but he's getting confused because so many people are only going conditionally. Help him determine if the party is happening or not.

INPUT:
The first line of input is T, the number of test cases that follow. The first line of each test case has two integers, N ($1 \leq N \leq 100$) and M ($0 \leq M \leq N$), which are the total number of members on the Programming Team and the minimum number of members who must attend the party for Dave to hold it, respectively.

On the next N lines are several space separated values. The first value on each line is the name of a member. The second value is one of three characters, "*g*", "*n*", or "*u*", which denote that this member is going, not going, or undecided, respectively. If the member is undecided, an integer D ($1 \leq D \leq N-1$) will follow, which denotes the number of members this person depends on to go to the party, followed by D space separated values, which are the names of those members.

Note: there will be no circular dependencies, meaning that it will always be possible to determine if the party is happening or not. Additionally, each person who is depended upon by another member will appear in the list of members.

OUTPUT:
For each test case, output a single line containing either "*YES*" or "*NO*", indicating if the party is happening or not.

[1] *Jotunheim* in Scandanavian mythology, is the home(land) of the Jotun (giants). Ergo, DaveHeim is the "home of the Dave" and by extension, has become the name of parties thrown at his place.

SAMPLE INPUT:	SAMPLE OUTPUT:
2 3 1 Baker g Ted u 1 Ron Ron n 5 5 Alex g Kyle g Naonao u 2 Alex Josh Will n Josh u 1 Kyle	YES NO

Problem G

PrimeDigit

Joon (a member of the UF Programming Team) loves prime numbers, so he devised a number system consisting of the first four prime numbers and zero. Hence, the number system consisting of 0, 2, 3, 5 and 7. Joon decides to call this number system PrimeDigit. Essentially, we are just working in base 5 instead of base 10. Base 5 only has the digits { 0, 1, 2, 3, 4 }. But in PrimeDigit we write { 0, 2, 3, 5, 7 }.

In base 10 to Base5 to PrimeDigit,

0 is written 0 is written 0
1 is written 1 is written 2
2 is written 2 is written 3
3 is written 3 is written 5
4 is written 4 is written 7
5 is written 10 is written 20
6 is written 11 is written 22

and so on.

Your goal is to perform basic operations, such as add, subtract, and multiply with the PrimeDigit number system.

INPUT:
The first line will begin with a number T, the number of test cases to follow.

Each test case consists of a single line, of the form

CMD A B

CMD is a command, and will always be one of ADD, SUB, MUL. A and B are numbers written in PrimeDigit.

Numbers in PrimeDigit may have more than 10 digits. However, you are guaranteed that for the given input the corresponding Base 10 number X is such that $-10^9 \leq X \leq 10^9$. A negative number in PrimeDigit is represented with a '-' preceeding it.

OUTPUT:
The output will consist of T lines, one for each of the preceding test cases.

SAMPLE INPUT:	SAMPLE OUTPUT:
11 ADD 2 2 ADD 3 3 ADD 5 5 ADD 2537 5732 SUB 32 23 MUL 57322 3372 ADD 2020 3030 SUB 75320 2357 ADD 2222 7777 MUL 2222 7777 MUL 235 727	3 7 22 20500 7 277753702 5050 72732 22220 22205557 225053

Explanation of “SUB 32 23”:

'32' in PrimeDigit is 21 in base 5. 21 in base 5 is 11 in base 10.
'23' in PrimeDigit is 12 in base 5. 12 in base 5 is 7 in base 10.
11 - 7 = 4, in base 10.
4 in base 10 is 4 in base 5.
4 in base 5 is '7' in PrimeDigit.

Problem H

Magic Van

The UF Programming Team is off to the ACM ICPC Southeast Regional programming contest, but there is a slight hiccup. Some people prefer to sit near other people, such as their favorite teammates.

Everyone can fit in the van (it is a very large capacity van), but we want to know if the van can accommodate everyone with their preferred seating arrangements. The large van does have limits though, as no person can sit near more than 4 other people in the van at the same time.

The preferred seating arrangements of a team member is simply a short list of other member's IDs that they want to sit next to. Knowing the preferred seating arrangement of all team members, determine if it is possible that everyone's preferences are satisfied. A seating arrangement doesn't satisfy everyone's preferences if there is at least one team member who isn't sitting next to all their preferred members.

Note: This relationship is symmetric: if Alex is sitting near Joon, then Joon is also sitting near Alex.

INPUT:
The first line will begin with a number T, the number of test cases to follow. For each test case, it begins with a single line containing a number N, the number of members on the team. Each member of the team has a unique ID from 1 to N.

The following N ($2 \leq N \leq 10^7$) lines will each begin with an integer K ($0 \leq K \leq 4$). The i-th line is the list of preferences for the team member with ID i. Following the number K, will be K numbers where each number is an ID of a favorite member.

It's possible that someone has a super favorite, and so they say the ID of the same member multiple times in their preferred seating arrangement.

You are guaranteed that a team member won't prefer sitting "next to himself or herself".

OUTPUT:
The output will consist of T lines, one for each of the given test cases. For each test case, you will output either "*YES*" or "*NO*", indicating whether it is possible to accommodate everyone's preferences in the given test case.

SAMPLE INPUT:	SAMPLE OUTPUT:
3 4 1 2 1 3 0 1 3 2 0 3 1 1 1 6 0 1 1 1 1 1 1 1 1 1 1	YES YES NO

Explanation of Test Case 1:

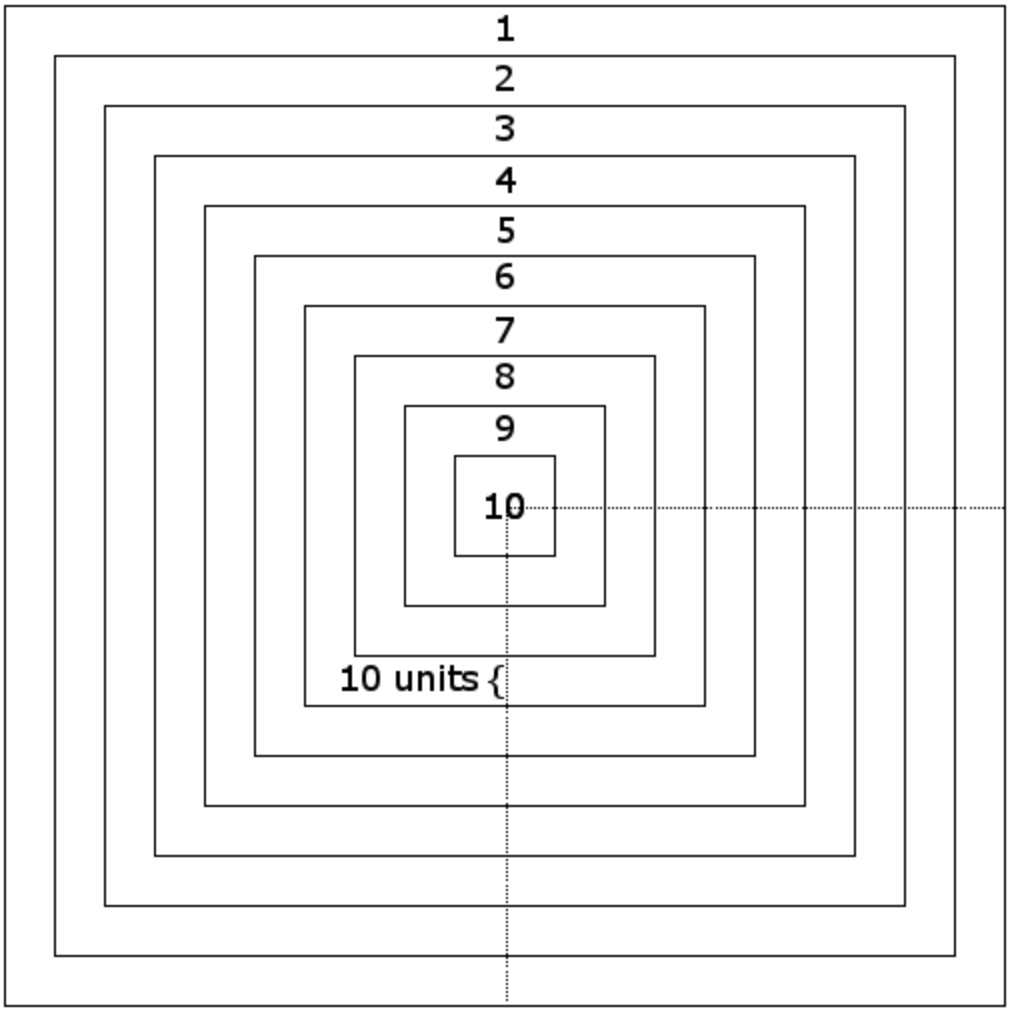
The following arrangement will satisfy everyone's preferences:
1 sits next to 2.
2 sits next to 1 and 3.
3 sits next to 2 and 4.
4 sits next to 3.

Problem I

Square Darts

The UF Programming Team is playing darts, but because we're unique and special and ***beautiful*** we're playing on a square board. Although we're unique and special and ***beautiful***, we're really lazy, so we need you to score our players' throws.

The dartboard is formed by exactly 10 concentric squares centered at the point (0,0) whose distance from one square's edge to the subsequent square's parallel edge is 10 units of distance. Each region for scoring is formed by the space between one square and the next largest subsequent square, with the exception of the centermost region which is formed only by the borders of the bullseye square. The centermost square for the bullseye is size 20x20. A diagram is provided below for visualization.



The regions of the dartboard from outward most to innermost score from 1 to 10 points. If a dart lands on the border of two different regions, we will score the throw as if it landed in the innermost region.

Each player is trying to score the most points possible by throwing at the bullseye region of the board. Each player will throw 10 darts at the board and calculate their score after they've thrown the tenth dart.

Given a series of 10 dart throws for each player as (x,y) coordinate locations on the board, print the player's resulting score. Each (x,y) coordinate will be a valid throw onto the board, i.e. the players will never miss a throw and throw off of the board entirely because they're pretty good at throwing darts at this weird square board.

INPUT:
The first line of input will have a single integer, N ($1 \leq N \leq 10^5$), the number of players we'll score. The subsequent $10N$ lines will contain each player's 10 dart throws. Each dart throw will be on its own line, represented by an integer x followed by a space followed by an integer y ($-100 \leq x,y \leq 100$).

OUTPUT:
For each player, print their resulting score on a single line.

SAMPLE INPUT:	**SAMPLE OUTPUT:**
2 10 10 20 20 30 30 40 40 50 50 60 60 70 70 80 80 90 90 100 100 13 14 18 43 42 66 77 33 45 45 90 95 50 50 34 39 87 78 62 10	55 48

Problem J

Work Splitter

The UF Programming Team has a squad of 2-5 members. Additionally, there are a set of problems to be solved. Each problem requires a certain amount of time (in minutes) to solve, and each problem needs to be assigned to one of the members to solve it. Some members may get more than one problem assigned, and some may get no task assigned at all (how lucky!).

All the members are equally competent (or incompetent), so regardless of who is assigned the problem, that problem will take the same amount of time to be completed. Unfortunately, the problems are somewhat complicated, so members can only work on one at a time and will not stop until the problem is solved.

As the coach, you need to consider the list of problems that need to be completed and decide how to assign them to your members. You are a kind coach, so you want to assign the problems in such a way so that the member who ends up needing to spend the most time to complete his tasks gets to finish as early as possible. Output the time that employee gets to leave.

INPUT:
The first line begins with a number T, the number of test cases to follow. Each test case begins with a pair of numbers, N and M, separated by a single space. N $(0 \leq N \leq 10)$ is the number of problems to solve, and M $(2 \leq M \leq 5)$ is the number of members on the squad. On the next line is a list of N numbers, separated by spaces. These numbers represent the time it takes to solve the problems, in minutes, where $1 \leq \textit{duration of problem} \leq 1000$.

OUTPUT:
The output will consist of T lines, one for each test case. Output the earliest that the last person could finish, if the problems are assigned optimally.

SAMPLE INPUT:	**SAMPLE OUTPUT:**
3 4 3 4 3 5 6 9 3 4 4 3 3 4 5 18 8 9 2 3 481 1	7 20 481

Problem K

Weight Balancing

William Macfarlane, of the UF Programming Team, is trying to set up his bench press so he can get swole. In order to get as big as possible, he will only lift with all of his weights on his bar. However he wants to set up the bar so that the sum of the weights on one side of the bar is equal to the sum of the weights on the other side of the bar.

Your job is to let William know if this is possible.

INPUT:
The first line begins with a number T, the number of test cases to follow. Each test case begins with a number N ($1 \leq N \leq 500$), the number of weights that William must try and balance. Following N will be N positive integers specifying the mass of William's weights, separated by spaces: $A_1 A_2 ... A_N$ ($1 \leq A_i \leq 200$).

OUTPUT:
The output will consist of T lines, one for each test case. Output "*YES*" if William can balance all of his weights, "*NO*" otherwise.

SAMPLE INPUT:	**SAMPLE OUTPUT:**
3 2 2 2 4 1 1 2 5 5 7 9 9 5 2	YES NO YES

Explanation of Test Case 2:

If Will only used the two weights of mass 1, the bar would be balanced. But Will won't accept a combination unless it uses all the available weights.