acm International Collegiate Programming Contest 2007 IBM. event sponsor

South Pacific Region

The 2007 problem set has benefited from contributions by many volunteers across Australia and New Zealand. I would like to thank Alex Flint, Alexandre Mah, Bernie Pope, Bradley Alexander, Clarence Dang, Eric McCreath, Michael J. Dinneen, Jimmy Foulds, Mark Tsui, and Paul Calder, Phil Robbins, Shawn Haggett, and Takeshi Matsumoto for ideas and solutions. I would also like to thank Chris Handley, Mike Cameron-Jones, and Robyn Gibson for careful reading of and suggesting improvements for the problem statements.

The 2007 problems are not meant to be in increasing order of difficulty (after all, difficulty is relative). However, I believe that the set includes three easy problems. One of these is "problem A". For the rest, happy hunting.

I hope that you enjoy the challenges this problem set will present you.

Problem A
Text Formatting

Your chief judge is getting long in the tooth and it is getting increasingly difficult for him to read through the densely-written messages of complaints about the problems and scores. The chief judge prefers to have more and uniform spacing between the words, and you have been drafted to write a program to format lines of text accordingly.

Your task is to write a program to read a number of lines of text and format each line independently such that:

1. successive words on a formatted line are separated by *exactly* two blank spaces, and
2. words are NOT split between lines, and
3. width of the formatted text does not exceed a specified value.

Leading and trailing blank spaces on each given line should be ignored.

As an example:

`  Your chief judge is geting long in the tooth.` `  Have a nice day.`

`Your  chief  judge`
`is  geting  long  in`
`the  tooth.`
`Have  a  nice  day.`

`01234567890123456789`

The first input line is formatted into three (3) lines of width bounded by 20 places, and words separated by two blank spaces in each line. The second input line is formatted independently on the fourth output line.

INPUT:

Input to this problem starts with an integer K, K > 0, that represents the number of messages. The number K is given on a separate line followed by a description of the K messages. The description of each message starts with a line that contains two integers. The first integer W, W ≥ 20, represents the desired width of the formatted text, and the second integer N, N > 0, represents the number of lines in the message. A single blank space separates the two integers. The message, which consists of N lines, follows with each line consisting of a sequence of one or more words separated by blank spaces. The length of each word is less than or equal to W. That is, a word like "supercalifragilistic-expialidocious" is only to be expected as part of the input if W ≥ 35.

OUTPUT:

For each message the output consists of one line. The line starts with the message number (the first message being "Message 1"), followed by a ": " (colon followed by space), as shown in the EXAMPLE OUTPUT below. This is followed by the number of lines that the formatted text would occupy.

EXAMPLE INPUT:

```
2
20 1
Your chief judge is geting long in the tooth.
30 2
For each message the output consists of one line.
The chief judge now prefers to have more and uniform spacing
```

EXAMPLE OUTPUT:

```
Message 1: 3
Message 2: 6
```

Problem B
Rating Points Exchange

Executive of the ACM (Australian Crikey Masters) club has recently decided to switch into a new rating system in their tournaments. The new rating system is based on a combination of match result, the players' rating, the pieces that remain on the board at the end of the game, as well as a handicap for the white player's advantage for playing first[1]. But it is complex and requires a lot of time to calculate in a tournament with a large number of participants. A task that must be completed between matches as the result of each match changes the players' ratings. The executive wants to speed-up the process of updating the players' ratings between matches through automation and you have been asked to implement this new rating system.

I shall now explain the updating process along with an example of a game between Alice and Bob, whose current club ratings happen to be 76.91 and 76.36 respectively in the following steps:

1. Alice, who is scheduled to play white, will be treated as though she is three (3) points stronger than her current rating. This gives Alice a rating of 79.91 against Bob's rating of 76.36, and thus a rating gap of 3.55 in favor of Alice. The rating gap is defined as the higher-rated player's rating minus the lower-rated player's rating.

2. Calculate core exchange CE as "rating gap / 10" (rounded to 2 decimals), which is 0.355 (rounded to 0.36) in Alice vs Bob game. In case you have forgotten, the digits 5 to 9 are rounded up and the digits 1 to 4 are rounded down.

3. Calculate the rating points exchange (RPE) based on the match result as follows:
   a. If the player with the higher rating (in this case, Alice) wins, then the RPE is calculated as "1 – CE". The winner's rating will go up by the RPE (in this case 0.64) and loser's rating down by the RPE.

[1] Just in case you are curious about the game of Crikey. It is a two player board game that uses black and white pieces. The list of rules is lengthy and sometimes vague. One day I shall list them.

b. If the player with the lower rating (in this case, Bob) wins, then the RPE is calculated as "1 + CE". The loser's rating will go down by the RPE (in this case 1.36) and the winner's rating will go up by the RPE.
c. If the game is drawn, then the RPE is calculated as "CE". The higher-rated player's rating (in this case Alice's rating) will go down by the RPE (in this case 0.36) and the lower-rated player's rating will go up by the RPE. If both players have the same rating (after applying the handicap rule), then their ratings will not change.

4. However if the winning side is left with crikey pieces of total value less than those of the losing side, then the RPE calculated above is then doubled; that is multiplied by 2. In our example, the RPE will be calculated as 2.72 (1.36 * 2) in the case of Bob winning the game with the value of his remaining pieces less than those of Alice. If the game is drawn, then this rule does not apply.

INPUT:

The first line of the input contains a single integer between 1 and 1000, inclusive, which is the number of tournaments that follow. The description of each tournament consists of "*n+m+1*" lines:

1. The 1st line consists of two integers *n* and *m* that identify the number of players and the number of games. The value of *n* is between 2 and 500, inclusive and the value of *m* is between 1 and 500, inclusive. The integers are separated by a single space.

2. Each of the following *n* lines consists of a string (with no blank spaces) that represents the player's name followed by a floating point number with 2 digits after the decimal point that represents the player's rating. The name of a player is a string of less than or equal to ten (10) lower-case letters.

3. Each of the next *m* lines contains a description of a single game in the form of three (3) strings followed by two (2) integers that are separated by single spaces. The first string is the name of the player playing white, the third string is the white player's result (lost,

drew, or won), and the first integer is the total value of the white player's pieces at the end of the game. The second string is the name of the player playing black, and the second integer is the total value of the black player's pieces at the end of the game.

OUTPUT:

For each tournament the output starts with a line that contains the tournament number (the first being "Ratings after Tournament 1"), followed by a ":", as shown in the EXAMPLE OUTPUT below, and then followed by a sequence of players' names and their ratings sorted in decreasing order of their final ratings. Players with the same rating must be sorted in decreasing alphabetic order.

EXAMPLE INPUT:	EXAMPLE OUTPUT:
4 2 1 alice 76.91 bob 76.36 alice bob lost 20 15 2 2 alice 55.55 bob 55.55 alice bob drew 10 30 bob alice drew 20 20 2 1 alice 94.26 bob 96.00 alice bob won 10 10 3 2 alice 52.55 bob 55.55 fred 80.00 alice bob drew 10 30 fred fred lost 27 50	Ratings after Tournament 1: bob 79.08 alice 74.19 Ratings after Tournament 2: alice 55.61 bob 55.49 Ratings after Tournament 3: bob 95.13 alice 95.13 Ratings after Tournament 4: fred 80.00 bob 55.55 alice 52.55

Problem C
Self-divisible Numbers

An integer number is said to be ***self-divisible*** if each digit divides the number formed by all digits up to and including that digit evenly. If you find yourself puzzled by the previous sentence, you should not worry. You are not alone. I needed to scribble few examples to understand it myself. Here are few of them:

1) 213 is a self-divisible number because
   2 divides 2, 1 divides 21, and 3 divides 213.
2) 201 is not a self-divisible number because
   2 divides 2, but 0 does not divide any number.
3) 2534 is not a self-divisible number because
   2 divides 2, 5 divides 25, but 3 does not divide 253.

Your task is to write a program to count the number of self-divisible numbers in a selected range. As an example, for the range of integers between 7 and 15, where

- 7, 8, 9, 11, 12, and 15 are self-divisible
- 10 is not self-divisible because it contains a zero, which cannot divide any number, and
- 13 and 14 are not self-divisible because the 2nd digit in each number does not divide it.

your program must calculate a count of 6.

INPUT:

Input to this problem starts with an integer N that represents the number of cases, N > 0, on a separate line followed by a description of the N cases. Each case is described on a separate line by two positive integers S and F, where S is less than or equal to F, that represent the start and finish values of a range of integers. The two numbers are separated by a single blank space as shown in the EXAMPLE INPUT below. Each of the integers S and F consists of K digits, 0 < K < 400, and the number of self-divisible numbers in the range between each pair is less than 1000000.

OUTPUT:

For each case, the output consists of one line. The line starts with the case number starting with the value of one (1), followed by a ": ", as shown in the EXAMPLE OUTPUT below, and then followed by the number of self-divisible numbers in the range.

EXAMPLE INPUT:

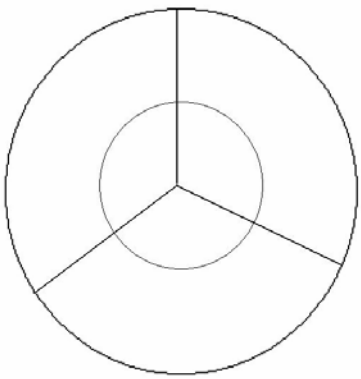
```
5
3 5
7 15
22222222222222222222 22222222222222222224
22222222222222222222 22222222222222222224
222222 2222222
```

EXAMPLE OUTPUT:

```
Case 1: 3
Case 2: 6
Case 3: 3
Case 4: 2
Case 5: 34865
```

Problem D
Challenging "*Butts*"

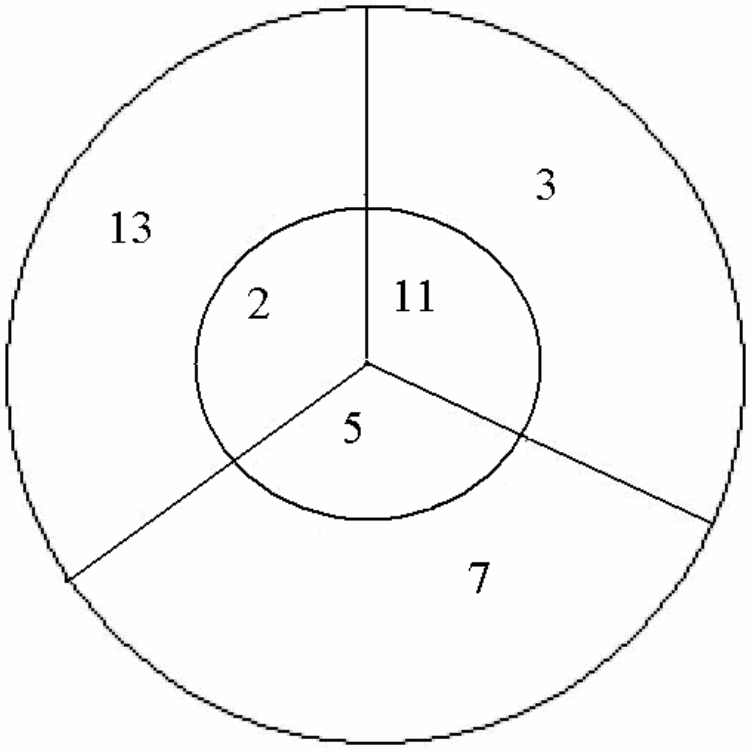
Darts[1] is a very popular game in which darts are thrown at a circular target (*dart board*) hung on a wall. Dart boards are usually made of sisal fibers or boar bristles, low quality boards are sometimes made of paper. A regulation board is 45.72 cm in diameter, and is divided into sectors. Each sector is lined with metal wire. The numbers indicating the various scores of sectors on the dart board are normally made of wire, especially on tournament-quality boards, but may be printed on the board instead. In the standard game, the dart board is hung so that the *bulls-eye* is 1.73 m from the floor, eye-level for a six foot person. The *oche*, the line behind which the player must stand, is 2.37 m from the face of the board. When playing darts players often aim at the high scoring sectors, but for ordinary players it is hard to land a dart on the desired sector. The risk of aiming at a sector can thus be measured by the difference between the scores of adjacent sectors, where two sectors are said to be adjacent if they share an edge or an arc. A large such difference increases the risk and makes the game more challenging. The total risk of a dart board is the sum over the risks between all adjacent sectors. We have been asked by the sponsor of a programming competition to design a new, and challenging, dart board to occupy the touchy coaches during the contest.



[1] Some historical records suggest that the first standard dartboards were the bottoms of wine casks, hence the game's original name of "butts".

The new dart board design consists of a circle that is divided into N sectors, N $\geq$ 3, by lines running from the centre of the circle to its perimeter and a smaller concentric circle that subdivides each sector into two areas: as shown in the sketch for N equal to three (3).

Your task is to write a program to read "2N" positive integer values and assign them to the "2N" areas of the new dart board design such that the total risk is maximized. An example of such an assignment is:



The total risk of this dart board design with "6" areas is 59.

INPUT:

The first line of the input contains a single integer between 1 and 1000, inclusive, which is the number of dart boards that follow. The description of each dart board consists of two lines:

1. The 1st line consists of an integer N, $300 \geq N \geq 3$, which identifies the number of sectors on the board.
2. The 2nd line consists of "2N" positive integers, separated by single spaces, which represent the scores. Each integer is less than or equal to 10000.

OUTPUT:

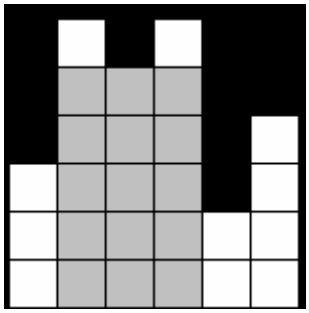
For each dart board the output is an integer, on a separate line, which represents the maximum risk of the board.

EXAMPLE INPUT	EXAMPLE OUTPUT
4 3 2 3 5 7 11 13 4 2 3 5 7 17 19 23 29 8 2 6 7 3 2 4 99 30 28 56 74 1 35 10 10 48 7 2 6 7 3 2 4 99 30 28 74 35 10 10 48	59 213 1035 870

Problem E
A Fitting Advertisement!

A skyline is the outline formed by a group of buildings against the sky. The ACM city in Second Life, which has been built with a beautiful skyline that's visible to anyone that approaches it, sold the rights to place advertisement at the approaches to the city to a company as long as it does so while preserving the shape of the city's skyline. The company wants to accept all requests for advertising that are of rectangular shape and can be fully placed, with sides parallel to edges of the skyline as shown shaded in the example below, within the interior of the skyline.

Each skyline of the ACM city is formed by N buildings, all with a width of one (1) but with different heights. The height of each building is between 0 and 1000, inclusive and the number of buildings N is between 1 and 400, inclusive. The example below shows a skyline with six (6) buildings, and an advertisement of size 3 by 5 (shown in gray) placed parallel to the sides of the skyline. Note that an advertisement may be rotated by 90^{o} so that it can fit into the skyline. That is, advertisement of size 5 by 3 can be placed within the skyline by rotating it first. It is "Second Life" after all.



Your task is to write a program to read the descriptions of a number of skylines and advertising requests, and decide for each request whether it should be accepted or rejected. Each request is to be checked independently as only one advertisement will be displayed at any time.

INPUT:

The first line of the input contains a single integer C between 1 and 1000, inclusive, which is the number of cases that follow. Each case starts with a description of the skyline of the city from one approach that consists of two lines:

1. The 1st line contains an integer N, $1 \leq N \leq 400$, that is the number of buildings in the skyline
2. The 2nd line contains the heights of the buildings

followed by a sequence of advertising requests. Each request consists of two integers on a single line, separated by a single space, which describe the length of the two sides of the rectangle that contains the advertisement. Both lengths are between 1 and 1000, inclusive. A request of two zeros separated by a single space (0 0) terminates the case.

OUTPUT:

For each skyline the output starts with a line that contains the case number starting with the value of one (1), followed by a ":", as shown in the EXAMPLE OUTPUT below, and then followed by a sequence of decisions of "Accept" or "Reject", on a separate line, for each request.

EXAMPLE INPUT:	EXAMPLE OUTPUT:
2 **6** **3 6 5 6 2 4** **3 5** **2 6** **8 1** **0 0** **7** **5 0 8 0 3 6 4** **3 4** **4 2** **0 0**	**Case 1:** **Accept** **Accept** **Reject** **Case 2:** **Reject** **Accept**

Problem F
The Zits Code

Jeremy is not known for his organizational skills, but he is now determined to change. Jeremy's plan is to place notes around the house to remind him of tasks to be done and of the proper ways to do them. Jeremy's plan also includes encrypting the messages so that his parents (*who don't understand anything*!) do not nag him, but in a simple way so that he can recover the original message easily.

Jeremy's encryption scheme consists of two steps:

1. Enter the message, of M characters, which includes the spaces between words, into a spiral that curls inwards in a clockwise direction, starting at the top-left corner of a square. The width and height of the square enclosing the spiral are chosen to be equal to the square root of P, where P is the smallest perfect square larger than or equal to M. If M is strictly less than P, then the remaining locations in the square are filled with the character '$'.
2. Write the encrypted note by writing the characters one row at a time starting with the top row.

As an example, for the following message of 33 characters

abcd fgh jklmn pqrstu wxyz1 34567

Jeremy writes the following note:

abcd ftu wxgs67$yhr5$$z q43 1jp nmlk

based on entering his original message in a square of 6 rows and 6 columns as follows:

a	b	c	d		f
t	u		w	x	g
s	6	7	$	y	h
r	5	$	$	z	
q	4	3		1	j
p		n	m	l	k

Your task is to write a program to encrypt Jeremy's messages with the hope that he will acquire some organizational skills, in peace. In case you have forgotten, let me remind you that a number X is a perfect square if there exists a positive integer K, such that K^2 equals X. For example, 16 is a perfect square but 18 is not a perfect square.

INPUT:

Input to this problem starts with an integer N that represents the number of messages, N > 0, on a separate line followed by a sequence of N messages. Each message consists of M characters, 1000 ≥ M > 0, on a single line with no blank spaces at the end.

OUTPUT:

For each message, the output consists of one line that contains the note with an encrypted message.

EXAMPLE INPUT:

```
5
Fold the clean laundry pile
make a decision about the may-be laundry pile
Make the dirty laundry pile invisible
01234567890123456789012345678901234 5
01234567890123456  890123456      012345
```

The last line has one blank space between 6&8 and three between 6 & 0

EXAMPLE OUTPUT:

```
Fold tdry phn$$$ieu$$$l a$$$ecl nael
make a he maydty pi-e r$$lbctd$$eeiunual soba noi
Make thpile ie $$$$n y$$$$vdr$$$$iidelbisrnual yt
01234590123681234770545869876954321 0
01234590123681234  705458      69543210
```

The last line has one blank space between 7&0 and three between 6 & 6

Problem G
DNA Chopper

As the sole employee in a pharmaceutical company with programming exposure, you have been asked to oversee the process of cutting DNA strands into smaller pieces, which has been outsourced to a company by the name of *Chopper*!, and check for any attempt of over charging.

DNA (**Deoxyribonucleic acid)** strand is a long polymer of simple units called nucleotides, with a backbone made of sugars and phosphate atoms joined by ester bonds. Attached to each sugar is one of four types of molecules called bases. It is the sequence of these four bases along the backbone that encodes information needed to construct other components such as proteins and RNA molecules. The cost of cutting a DNA strand, with today's technology, is equal to the length of the strand, where a "cut" refers to splitting a strand into two pieces. The cost of a single cut does not depend on the location where the cut is made. *Chopper*! claims to have developed the world-best-practice in sequencing the cuts of a DNA strand and to deliver the most savings to its customers.

Your task is to write a program to process the next batch of DNA strands and calculate the best price for slicing each of them into smaller strands of integer lengths. For each DNA strand you are given its length as an integer N, $N \leq 10000$, and a list of M, $2 \leq M \leq 15$, strands' lengths to be produced. The sum of the M integers equals the original strand length N.

The total cost of slicing a given strand depends on the choice of locations and the order of the M-1 required cuts. As an example, for a given list of M (=4) lengths 2, 2, 5 and 5 and a DNA strand of length 14, we can:

- first cut the strand in half then twice split a 2 and 5 from the remaining 7s for a cost of 28 (=14 + 7 + 7), or
- first cut off a 5 from the end of the strand, then another 5, then split the remaining 4 in half for a cost of 27 (= 14 + 9 + 4).

INPUT:

The input contains information about a number of DNA strands to be processed. The information for each strand consists of two lines:

1. The 1st line consists of the two integers M and N ($2 \leq M \leq 15$ and $0 \leq N \leq 10000$) that identify the number of required smaller strands and the length of DNA strand, respectively.
2. The 2nd consists of M positive integers (separated by blank spaces) that sum to N.

The input is terminated by a line of two zeros (0 0) for which no output is to be produced.

OUTPUT:

For each input DNA strand the output is an integer, on a separate line, which represents the minimum cost for slicing the strand into the given list.

EXAMPLE INPUT:

```
4 14
2 5 2 5
5 32
3 3 5 5 16
0 0
```

EXAMPLE OUTPUT:

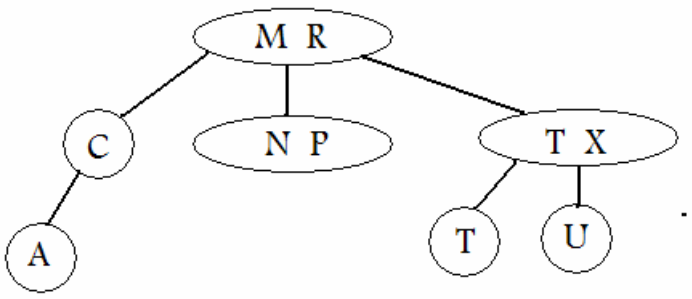
```
27
64
```

Problem H
Automatic Marking

You must be familiar with binary trees and all their operations, but this problem deals with a less popular structure, which I shall call *myTree*. There are three possible organizations of a *myTree*:

1. an empty tree. That is, a tree with no nodes.
2. a tree whose root node has a single data item, say K, and two children. Each of its two children is a *myTree*. Any values in the left subtree are less than or equal to K, and any values in the right subtree are larger than K.
3. a tree whose root node has two data items, say K_1 and K_2 , and three children. $K_1 < K_2$. Each of its three children is a *myTree*. Any values in the left subtree are less than or equal to K_1, any values in the middle subtree are larger than K_1 and smaller than or equal to K_2 , and any values in the right subtree are larger than K_2.

All internal (non-terminal) nodes have two or three children, although some may be empty. One way to represent such a tree is to use level-order traversal, starting at the root node, with the content of each node enclosed in parentheses. An empty tree is represented by a pair of parentheses that encloses nothing. The following figure demonstrates an example of such a tree, along with its representation, with values in the nodes being uppercase characters chosen in the range of "A" to "Z", inclusive.



(M R) (C) (N P) (T X) (A) () () () () (T) (U) () () () () () () ()

A lecturer of "Data Structures 101" likes to test her students understanding of the *myTree* structure by asking them to identify all possible ways to assign a missing value in a given *myTree*.

Examples of such a question would be:

1. a tree "(M R) (C) (N P) (? U) () () () () () () () ()", for which the answer should be the letters "S" and "T".
2. a tree "(M R) (X) (N O) (? U) () () () () () () () ()", for which the answer should be "This is not a *myTree*". The reason is that X > M.
3. a tree "(M R) (N O) (? U)", for which the answer should be "This is not a myTree". The reason is that nodes with two values should have three children in a *myTree* structure, which is violated in this question.

Your task is to write a program to answer such a question.

INPUT:

The input contains descriptions of a number of *myTree structures* to be processed. The information for each tree is given in a single line as a series of properly matched parentheses. Each pair of matched parentheses encloses zero, one, or two characters selected from uppercase letters in the range of "A" to "Z" and "?". Each line contains exactly one "?". The selected characters and parentheses are separated by single spaces.

The input is terminated by a line of a set of matched parentheses, which encloses a zero, for which no output is to be produced.

OUTPUT:

For each input tree the output is a single line with:

1. a listing of possible uppercase letters sorted in increasing order that can replace the "?" in the given tree, or
2. the string "This is not a myTree", if the given data does not conform to the given specifications.

EXAMPLE INPUT:

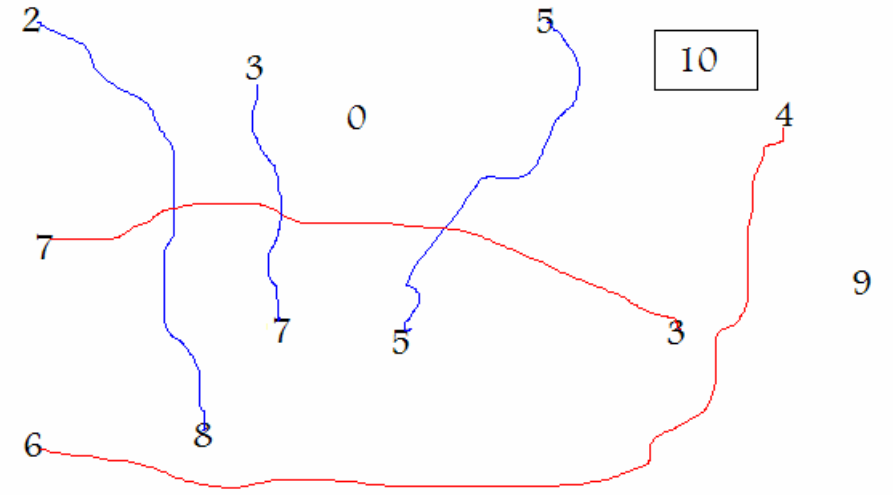
```
(M R) (C) (N P) (? U) () () () () () () () ()
(M R) (X) (N P) (? U)
(M R) (C) (? U)
(M R) (C) (N P) (? X) (A) () () () () (S) (U) () () () () () () ()
(0)
```

EXAMPLE OUTPUT:

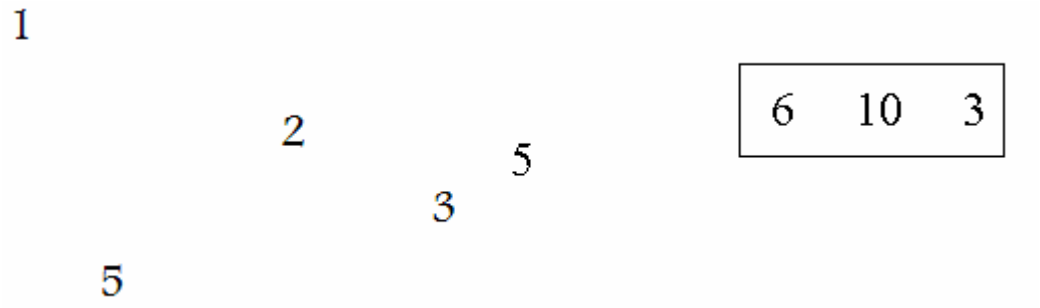
```
S T
This is not a myTree
This is not a myTree
S T
```

Problem I
Mr. Thompson's Problem

Mr. Thompson keeps the pupils in his class well behaved by dishing out a challenging class-quiz when they get rowdy. The quizzes are usually created as a variation of a familiar problem, cunning eh! He has recently used a typical 1st grade exercise in which pupils are given a sheet of paper with numbers scattered on it and a special number inside a box, as shown,



and are asked to connect pairs whose sum is equal to the number in the box with each number outside the box to be used at most once. A possible perfect answer is shown in colors. Mr. Thompson's quiz looks similar except that a pair of numbers can be connected if their sum equals any of the numbers in the box,



and the challenge is to find the largest number of possible connections. A pupil who rushes to connect "1 to 5" will miss the chance to obtain two (2) pairs by connecting "5 to 5" and "1 to 2", which is the largest possible.

Mr. Thompson wants to have a large number of these quizzes, but he does not have the time to find the right answer for each one. Your task, as an admirer of your primary school teacher, is to help by writing a program to calculate the largest number of possible connections for him. The input to your program consists of a collection **C** of *n* integers, and a set **S** of *m* integers. A connection between elements in **C** is valid if the sum of the corresponding integers is an element in **S** with the constraint that each element in **C** may be used at most once. A value in **S** may be used more than once.

INPUT:

The first line of the input contains a single integer between 1 and 100, inclusive, which is the number of problems that follows. Each problem description consists of three lines: The 1st line contains two integers, *n* and *m* that represent the sizes of the list **C** and the set **S**, respectively. The 2nd line contains the *n* integers of **C**, and the 3rd line contains the *m* integers of **S**. The values in each line are separated by single spaces. $2 \leq n \leq 200$, and $1 \leq m \leq 100$.

OUTPUT:

For each problem, the output is a single line consisting of an integer that is the maximum number of possible pairings.

EXAMPLE INPUT	EXAMPLE OUTPUT
2 6 4 1 2 3 4 4 5 6 9 3 5 5 4 1 2 3 4 5 6 9 3 5	3 2