TCEA
HIGH SCHOOL
PROGRAMMING CONTEST

AREA PROBLEM SET
FEBRUARY 2009

Texas Computer Education Association
2009 High School Programming Contest
Area Problem Set

Problem 2.1 Making Money

General Statement: A small business owner team wants to track the amount of money made each day. At the start of the day, the owner counts the money contained in the register and records it. At the end of the day, the owner again counts the money contained in the register. The owner knows that the amount of money made during the day is the difference between the second total and the first total. Your task is to write a program that computes this difference.

Input: The first line of the input contains an integer *n* that represents the number of data collections that follow where each data collection is on a single line. Each data collection contains two integers separated by a single space. The first integer represents the total amount of money in the register at the start of the day. The second integer represents the total amount of money in the register at the end of the day.

Name of Data File: pr21.dat

Output: Your program should produce *n* lines of output (one for each data collection). The output for each data collection should contain a single integer representing the difference between the second amount and the first amount.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: The value of *n* will be between 1 and 1000, inclusive.
Each input integer will be between 0 and 1,000,000, inclusive.
In each data collection, the final amount will not be smaller than the starting amount.

Sample Input:

```
3
5 505
755 755
1 2000
```

Sample Output:

```
500
0
1999
```

Problem 2.2 Adding Articles

General Statement: When using the article “a” before a singular noun, we must first check whether the noun starts with a consonant or a vowel. If the noun starts with a consonant, then we can use the article a. If the noun starts with a vowel, then we should use the article `an`. For this problem, assume that this rule is always true.

Given a set of nouns, your task is to write a program that will determine if each noun should use a or `an`.

Input: The first line of the input contains an integer *n* that represents the number of data collections that follow where each data collection is on a single line. Each data collection contains a single noun containing only lower case letters (no spaces or punctuation characters).

Name of Data File: pr22.dat

Output: Your program should produce *n* lines of output (one for each data collection). The output for each data collection should contain the input noun preceded by the appropriate article (either a or an) depending on whether the first letter of the noun is a vowel. Both the article and the noun should be displayed using only lower case letters. There should be a single space between the article and the noun.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: The value of *n* will be between 1 and 1000, inclusive.
Each input noun will have between 2 and 80 letters, inclusive.
The only vowels are a, e, `i`, o, and u.

Sample Input:

```
3
cat
apple
year
```

Sample Output:

```
a cat
an apple
a year
```

Problem 2.3 Trend Tracking

General Statement: A school wants to determine if a trend can be detected in their students' standardized test scores over the last three years. Given the last three test scores of a student, they would like to determine if the scores are increasing or decreasing. Your task is to write a program that can perform this function.

Input: The first line of the input contains an integer *n* that represents the number of data collections that follow where each data collection is on a single line. Each data collection contains three integers, and there is a single space between each pair of integers. Each of these integers represents a student's test score.

Name of Data File: pr23.dat

Output: Your program should produce *n* lines of output (one for each data collection). The output for each data collection should indicate whether the trend of the scores is increasing or decreasing. If the trend is increasing, your program should print `INCREASING`. If the trend is decreasing, your program should print `DECREASING`. If the trend is neither increasing nor decreasing, your program should print `NO TREND`.

For the trend to be increasing, the second value must be greater than or equal to the first value, the third value must be greater than or equal to the second value, and the third value must be greater than the first value (not equal to). For the trend to be decreasing, the second value must be less than or equal to the first value, the third value must be less than or equal to the second value, and the third value must be less than the first value (not equal to).

The output is to be formatted exactly like that for the sample output given below.

Assumptions: The value of *n* will be between 1 and 1000, inclusive.
Each input integer will be between 0 and 100, inclusive.

Sample Input:

```
4
75 80 90
51 50 50
77 79 78
100 100 100
```

Sample Output:

```
INCREASING
DECREASING
NO TREND
NO TREND
```

Problem 2.4 Star Strings

General Statement: A Web site is compiling consumer reviews of products in order to produce an overall summary. Each reviewer describes why he or she liked or disliked the product and submits it to the Web site. At the end of the description, the reviewer indicates how many stars he or she believes the product deserved. This is indicated as a string of either one star (*), two stars (**), three stars (***), or four stars (****). The one and two star strings indicate a negative review, and the three and four star strings indicate a positive review.

Your task is to write a program that counts the number of positive and negative reviews of a product given the collection of star strings extracted from its reviews.

Input: The first line of the input contains an integer *n* that represents the number of data collections that follow where each data collection is on a single line. Each data collection contains a collection of star strings. Each pair of star strings is separated by a single space.

Name of Data File: pr24.dat

Output: Your program should produce *n* lines of output (one for each data collection). The output for each data collection should contain two integers separated by a single space. The first integer should be the number of positive reviews (three or four star strings). The second integer should be the number of negative reviews (one or two star strings).

The output is to be formatted exactly like that for the sample output given below.

Assumptions: The value of *n* will be between 1 and 1000, inclusive.
The number of strings in each data collection will be between 1 and 80, inclusive.
All input star strings will have between 1 and 4 stars, inclusive.

Sample Input:

```
3
*** **** *
* * * * * * * * * *
****
```

Sample Output:

```
2 1
0 10
1 0
```

Problem 2.5 Hidden Hypotenuses

General Statement: If *a*, *b*, and *c* represent the lengths of the sides of a right triangle, and *c* specifically represents the length of the hypotenuse, then we know that $c^2 = a^2 + b^2$. With this formula, it is possible to determine if any given set of three integers can represent the lengths of the sides of a right triangle. Your task is to write a program that can perform this function.

Input: The first line of the input contains an integer *n* that represents the number of data collections that follow where each data collection is on a single line. Each data collection contains three positive integers with a single space between each pair. Note that any of the three integers could represent the triangle's hypotenuse.

Name of Data File: pr25.dat

Output: Your program should produce *n* lines of output (one for each data collection). On each output line, if the integers in the corresponding input collection can represent the lengths of a right triangle's sides, your program should print `RIGHT`. If they cannot represent the lengths of a right triangle's sides, your program should print `NOT RIGHT`.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: The value of *n* will be between 1 and 1000, inclusive.
Each input integer will be between 1 and 1000, inclusive.

Sample Input:

```
3
4 5 3
20 10 10
10 6 8
```

Sample Output:

```
RIGHT
NOT RIGHT
RIGHT
```

Problem 2.6 Draw Four

General Statement: In the game Uno, players take turns placing cards on top of a pile. Each card has a color and a number. On each turn, a player must play a card that matches the top card's number or one that matches its color. So, if the top card is red and has the number 6, the player must play a red card or a card with the number 6. No other cards would be legal. Note that this problem is not considering "Wild Cards" that actually exist in the game.

Your task is to write a program that can determine if an input card can legally be played given the card at the top of the pile.

Input: The first line of the input contains an integer *n* that represents the number of data collections that follow where each data collection is on a single line. Each data collection contains two strings separated by a single space. Each string represents a card and contains two characters. The first character is a capital letter representing the color of a card, and the second character is a digit representing the number of a card.

Name of Data File: pr26.dat

Output: Your program should produce *n* lines of output (one for each data collection). The output for each data collection should indicate whether the first card could be played on top of the second card. If the colors or numbers match, the output should be `LEGAL`. If both the colors and numbers do not match, the output should be `ILLEGAL`.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: The value of *n* will be between 1 and 1000, inclusive.
In each input string, the first character will either be `B` (for Blue), `R` (for Red), `G` (for Green), or `Y` (for Yellow).
In each input string, the second character will be a digit from 0 to 9.
No other Uno cards are considered in the problem (no Skips, Draw-Twos, Reverses or Wild Cards).

Sample Input:

```
3
R6 R0
B3 G4
Y9 B9
```

Sample Output:

```
LEGAL
ILLEGAL
LEGAL
```

Problem 5.1 Coding Conundrum

General Statement: When sending a coded message to a friend, a basic encryption method is to simply encode each letter of the message by the number corresponding to its order in the alphabet (`A` is encoded as `1`, `B` is encoded as `2`, … , `Z` is encoded as `26`). Spaces in the message are encoded as `0`. Thus, the phrase `MY CAT` would be encoded as `13 25 0 3 1 20`. Unfortunately, this method is almost everyone's first impulse, so almost everyone would be able to break it easily.

To make the scheme harder, one approach is to encode each letter with some number that, when divided by 27, gives a remainder equal to the letter's order in the alphabet. For example, `A` could be encoded by `1, 28,` or `55` since each of these numbers gives a remainder of 1 when divided by 27. In general, `A` could be encoded by any number of the form *27k+1* where *k* is a nonnegative integer. Similarly, `B` could be encoded by any number of the form *27k+2*, `C` could be encoded by any number of the form *27k+3*, and so on. Spaces could be encoded by any number of the form *27k+0*.

Your task is to write a program that will decode a given message that was encrypted using this scheme.

Input: The first line of the input contains an integer *n* that represents the number of data collections that follow where each data collection is on a single line. Each data collection represents a message encoded using the above scheme and contains a collection of nonnegative integers with each pair of integers separated by a single space.

Name of Data File: pr51.dat

Output: Your program should produce *n* lines of output (one for each data collection). The output for each data collection should contain the decoded message using only spaces and capital letters.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: The value of *n* will be between 1 and 1000, inclusive.
Each input collection will contain between 1 and 100 integers, inclusive.
All input integers will be between 0 and 1,000,000, inclusive.
Each input integer represents a capital letter or a space.
There will be no punctuation symbols in the input.

Problem 5.1 Coding Conundrum (page 2 of 2)

Sample Input:

```
3
283 25 729 1110 55 2693
20 57 32 82 270 136 126 5 28 0 30 69 68 74 86 100 47
2724 3022 26 210005 810025
```

Sample Output:

```
MY CAT
TCEA AREA CONTEST
XYZZY
```

Problem 5.2 Maintaining Membership

General Statement: An organization periodically produces a report listing its current membership. During the time period between reports, current members will leave and new ones will join. So, two successive reports can be analyzed to determine which members are new and which members have left. Your task is to write a program that can perform this analysis.

Input: The first line of the input contains an integer *n* that represents the number of data collections that follow where each data collection is on a single line. Each data collection contains two strings representing the old and new reports, respectively. Each report string consists of a collection of capital letters with each letter uniquely representing a member. Thus, a report `ABC` means that `A`, `B`, and `C` are currently members. The letters in each report string will be given in alphabetical order (ascending).

Name of Data File: pr52.dat

Output: Your program should produce *n* lines of output (one for each data collection). Each output line should list the members that are new followed by the members that have left. There should be a single space between the two lists. Each list should be formatted as the input report, meaning that each list will contain a collection of capital letters with each letter corresponding to a member. The capital letters in each list should be given in ascending alphabetical order.

If either the list of new members or the list of members that have left is empty, your program should print `none` (in lowercase letters) for the list.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: The value of *n* will be between 1 and 1000, inclusive.
Each report string will have between 1 and 26 letters, inclusive.
Each letter in a single report string will be unique.

Sample Input:

```
3
AEIOU BCEISTU
WXYZ Y
ABCDEFGHIJKLMNOPQRSTUVWXYZ ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Sample Output:

```
BCST AO
none WXZ
none none
```

Problem 5.3 Rise over Run

(page 1 of 2)

General Statement: An equation of a line in the form $y = mx + b$ can be determined from two input points (x_1, y_1) and (x_2, y_2). The slope, m, can be found by dividing the difference in the y-coordinate values ($y_2 - y_1$) by the difference in the x-coordinate values ($x_2 - x_1$). Once m has been determined, the y-intercept, b, can be found by plugging in one of the input points in to the equation and solving it (for b). Thus, you could use either equation $b = y_1 - mx_1$ or $b = y_2 - mx_2$.

Your task is to write a program that will produce the equation of a line given two input points (x_1, y_1) and (x_2, y_2).

Input: The first line of the input contains an integer n that represents the number of data collections that follow where each data collection is on a single line. Each data collection contains four integers x_1, y_1, x_2, and y_2, with a single space between each pair. The first two integers represent the x-coordinate and y-coordinate of the first point, respectively. The second two integers represent the x-coordinate and y-coordinate of the second point, respectively.

Name of Data File: pr53.dat

Output: Your program should produce n lines of output (one for each data collection). The output for each data collection should contain a string in the following format `y=mx+b`. There should not be any spaces in the output string. The `y`, equal sign (`=`), and `x` should be displayed literally.

The value for `m` should be an integer equal to the slope of the line indicated by the input points. This integer should always be displayed even if it is equal to 1, 0, or -1. The value for `+b` should be a signed integer equal to the line's y-intercept. If `b` is greater than or equal to 0, its value should be preceded by a plus sign (`+`). If `b` is negative, its value should be preceded by a minus sign (`-`).

The output is to be formatted exactly like that for the sample output given below.

Assumptions: The value of n will be between 1 and 1000, inclusive.
The slope of the resulting line will always be an integer.
The value of each coordinate point will be between -1,000,000 and 1,000,000, inclusive.
x_1 will not be equal to x_2.

Problem 5.3 Rise over Run (page 2 of 2)

Sample Input:

```
4
1 3 5 27
1000 0 -1000 2000
5 5 -3 -3
10 -1 -10000 -1
```

Sample Output:

```
y=6x-3
y=-1x+1000
y=1x+0
y=0x-1
```

Problem 5.4 A-Team

General Statement: During Physical Education class, the instructor groups the students into *k* teams, for some number *k*. The grouping is accomplished by placing all of the students in a line and having them call out numbers. The person at the far left calls out 1, the person next to him or her calls out 2, the next person calls out 3, and so on. This continues until a person calls out the number *k*. At this point, the next person will restart the counting beginning from 1 again. This whole process continues until the person at the far right calls out a number.

To form teams using this process, the instructor groups students that called the same number. So, Team #1 is formed from all of the students that called out number 1. This is true for any number from 1 to k.

To illustrate the above example, consider grouping 10 students into 4 teams. The first four students would call out 1, 2, 3, and 4, respectively. At this point, the counting restarts, so the fifth, sixth, seventh, and eighth students would call out 1, 2, 3, and 4, respectively. Finally, the ninth student would call out 1, and the tenth student would call out 2. Thus, Team 1 would have the first, fifth, and ninth students.

A Student, which we will call Student A, wants to figure out his or her team members before the counting begins. A realizes that all that is needed is to identify the position number of all of the students and divide it by *k*. Those whose remainders have the same remainder as Student A will be on A's team. Your task is to write a program that will accomplish this.

Input: The first line of the input contains an integer *n* that represents the number of data collections that follow where each data collection is on a single line. Each data collection contains two components separated by a single space. The first is an integer, *k*, representing the number of teams to create. The second is a string of capital letters where each letter represents a student. Only one student will be represented by the letter A. Other letters may be duplicated.

Name of Data File: pr54.dat

Output: Your program should produce *n* lines of output (one for each data collection). Each line should contain the capital letters representing the students on A's team. You should not include the letter A in the output (Student A only wants to know his or her teammates). The output letters should be sorted alphabetically in ascending order.

The output is to be formatted exactly like that for the sample output given below.

Problem 5.4 A-Team (page 2 of 2)

Assumptions:

The value of *n* will be between 1 and 1000, inclusive.
The value of each *k* will be between 1 and 100, inclusive.
The length of each input string will be between 1 and 100, inclusive.
There will only be one A in each input string.

Sample Input:

```
3
4 WXYZABCDEF
2 THISISANINPUTSTRING
1 ZXYWWXYZA
```

Sample Output:

```
EW
GIIIIPTTT
WWXXYYZZ
```

Problem 5.5 Next Unique Number

General Statement: There is a difference between numbers such as `1234`, `743`, `15`, and `16490`, and other numbers such as `733`, `4041`, `10078`, and `505`. All of the digits in the first set of numbers are unique meaning that each digit in the first set of numbers only appears once in its number. This is in contrast to the second set of numbers where each one has at least one digit that appears twice.

Your task is to write a program that inputs an integer, *x*, and identifies the smallest integer greater than or equal to *x* whose digits are all unique.

Input: The first line of the input contains an integer *n* that represents the number of data collections that follow where each data collection is on a single line. Each data collection contains a single integer, *x*.

Name of Data File: pr55.dat

Output: Your program should produce *n* lines of output (one for each data collection). The output for each data collection should be the smallest integer greater than or equal to *x* whose digits are unique.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: The value of *n* will be between 1 and 100, inclusive.
The value of each input *x* will be between 1 and 1,000,000, inclusive.

Sample Input:

```
3
550
988
54321
```

Sample Output:

```
560
1023
54321
```

Problem 5.6 MasterMind

General Statement: In one variant of the game MasterMind, a player selects four colored pegs and places them in some order. This represents the solution of the game, and both the pegs selected and their ordering are hidden from a second player. The purpose of the game is for the second player to guess the ordering.

A guess is made by specifying four colored pegs. The first player evaluates the guess by telling the second player two values. The first value represents the number of colored pegs in the guess that are in the solution and in the correct position. The second value represents the number of colored pegs in the guess that are in the solution but are in the wrong position. The second player then makes another guess which is subsequently evaluated as well. This process continues until the second player guesses the solution correctly.

Your task is to write a program that will evaluate a player's guesses given a solution to a MasterMind game.

Input: The first line of the input contains an integer *n* that represents the number of guesses to evaluate. The next line contains a single string that represents the solution of the MasterMind game. The solution is represented as 4 capital letters with each letter representing a colored peg.

The following *n* lines represent the guesses with each guess on a single line. Like the solution, each guess is represented as a single string with four capital letters with each letter representing a colored peg.

Name of Data File: pr56.dat

Output: Your program should produce *n* lines of output (one for each guess). The output for each guess should contain two integers separated by a single space. The first integer should be the number of pegs in the corresponding guess that are in the correct position. The second integer should be the number of pegs in the corresponding guess that are in the solution but in the wrong order.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: The value of *n* will be between 1 and 100, inclusive.
Each color is represented by a unique capital letter.
There could be more than one peg of the same color.

Problem 5.6 MasterMind (page 2 of 2)

Sample Input:

```
5
RGGB
BBBG
BBGG
RYYY
RGBG
RGGB
```

Sample Output:

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

Problem 9.1 Letters Left (page 1 of 2)

General Statement: In a Word Search, the player searches for a list of words within a two-dimensional grid of letters. Each word may appear in the grid displayed forwards, backwards, upwards, downwards, or in a diagonal line. When a word is found in the grid, the player usually marks its letters to indicate they have been used. Note that even if a letter is marked as part of a word, it may still be used as part of a different word.

After all of the words in the word list have been found, some of the letters in the grid will have never been used. When read in order from the top line to the bottom line scanning from left to right on each line (row-major order), these unused letters form a word representing the ultimate solution to the Word Search. Given a Word Search puzzle, your task is to find its ultimate solution.

Input: The first line of the input contains three integers with each pair of integers separated by a single space. The first integer, *r*, represents the number of rows in the grid of letters. The second integer, *c*, represents the number of columns in the grid of letters. The third integer, *n*, represents the number of words in the word list.

The next *r* lines of the input represent the grid of letters where each line contains *c* characters. Each of these characters is an uppercase letter. The last *n* lines of input contain the word list with each word on a single line. Each word contains only uppercase letters.

Name of Data File: pr91.dat

Output: Your program should produce one line of output which contains the ultimate solution of the Word Search. The ultimate solution is formed from outputting the unused letters in row-major order. The output should be displayed using only capital letters.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: The value of *r* will be between 1 and 30, inclusive.
The value of *c* will be between 1 and 30, inclusive.
The value of *n* will be between 1 and 100, inclusive.
The Ultimate Solution will contain at least one letter.
Each word in the word list will have between 2 and 30 letters, inclusive.
Each word in the word list will appear in the grid exactly once.

Problem 9.1 Letters Left

Sample Input:

```
4 6 6
CDORBA
YELLOW
ARUONG
NEERGE
GREEN
RED
YELLOW
BLUE
GOLD
CYAN
```

Sample Output:

```
ORANGE
```

Problem 9.2 Simplifying Sums

General Statement: Given two fractions with each fraction expressed in the form *a/b*, a program can be written that produces their sum as a fraction also in the form *a/b*. The sum should be reduced to its simplest form. Your task is to write this program.

Input: The first line of the input contains an integer *n* that represents the number of data collections that follow where each data collection is on a single line. Each line contains two fractions separated by a single space. Each fraction is in the form *a*/*b* where *a* and *b* are integers.

Name of Data File: pr92.dat

Output: Your program should produce *n* lines of output (one for each data collection). The output for each data collection should contain a fraction in the form of *a/b* where *a* and *b* are integers. The fraction should be reduced to its simplest form. Note that if *a* is evenly divisible by *b*, then *b* should be reduced to 1 (such as 1/1, 2/1, 3/1, etc.). Any sum that is equal to 0 should be expressed as 0/1.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: The value of *n* will be between 1 and 100, inclusive.
Each input integer *a* will be between 0 and 1,000,000, inclusive.
Each input integer *b* will be between 1 and 1,000,000, inclusive.

Sample Input:

```
5
1/2 3/4
33/24 1/8
7/12 5/12
0/100 8/2
0/100 0/10
```

Sample Output:

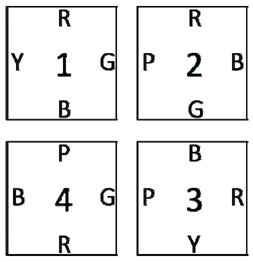
```
5/4
3/2
1/1
4/1
0/1
```

Problem 9.3 Four Square (page 1 of 2)

General Statement: The famous adventurer Indiana Smith has encountered a series of unusual locks that need to be opened in order to advance. Each individual lock consists of a series of four squares arranged in a 2x2 pattern meaning that each square borders two other squares in the lock.

After inspecting the squares, Smith has noticed that each one has a colored strip on each of its sides. Also, Smith noticed that each square can turn in a clockwise manner. From this, Smith concluded that to release a lock, it is necessary to repeatedly turn its four squares so that the borders of each pair of adjacent squares match. Note that in this problem, a turn means a 90° rotation clockwise.

To illustrate, consider the following figure which contains an example of a lock. Let the four squares of the lock be numbered from 1 to 4 in clockwise order as shown in the figure. Each of the borders is marked with a letter representing its color (R for Red, G for Green, B for Blue, Y for Yellow, and P for Purple). If the 2nd and 4th squares are each rotated once, then the borders between the 1st and 2nd squares would both be (G)reen, the borders between the 2nd and 3rd squares would both be (B)lue, the borders between the 3rd and 4th squares would both be (P)urple, and the borders between the 4th and 1st squares would both be (B)lue. So, this lock only requires two (2) total turns to release it.



Your task is to write a program that will determine the minimum number of turns needed in order to release a lock given its border colors.

Input: The first line of the input contains an integer *n* that represents the number of data collections that follow where each data collection is on a single line. Each data collection contains the information for a single lock and consists of four strings with a single space between each pair of strings. Each string represents one of the squares of the lock where the ith string represents the ith square in the above figure. Each string contains four characters *NESW*. These characters represent the colors of the (N)orth, (E)ast, (S)outh, and (W)est borders, respectively. Thus, the lock in the above figure would be given as `RGBY RBGP BRYP PGRB`.

Problem 9.3 Four Square (page 2 of 2)

Name of Data File: pr93.dat

Output: Your program should produce *n* lines of output. Each output line should contain a single integer representing the minimum number of clockwise rotations needed to release the lock.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: The value of *n* will be between 1 and 100, inclusive.
Each lock will have a solution.
Each input border color will be represented as a single uppercase letter.

Sample Input:

```
3
RGBY RBGP BRYP PGRB
OBGP KWPY RWGS ORTC
RGRG BYGG GOKW RWBY
```

Sample Output:

```
2
6
0
```

Problem 9.4 Seating Solutions

General Statement: A local organization is holding its annual banquet next month. The audience attending the event will be seated around several tables, and each table can seat up to 8 people. Tickets are sold so that a group of people can all request to be seated together. Thus, if a group of 5 people make such a request, they must all be seated at the same table.

The group purchases pose a dilemma for the banquet organizers in that they want to satisfy all group requests, but still would like to have as few tables as possible to seat everyone. For example, if 15 people purchase tickets individually, the organizers only need to set up 2 tables to seat them all. However, if three groups with 5 people in each group purchase tickets, the organizers must set up 3 tables.

Your task is to write a program that will determine the minimum number of tables required to satisfy a set of group seating requests.

Input: The first line of the input contains an integer *n* that represents the number of data collections that follow where each data collection is on a single line. Each data collection contains a collection of integers with a single space between each pair. Each integer represents the number of people in a group making a request to be seated together.

Name of Data File: pr94.dat

Output: Your program should produce *n* lines of output (one for each data collection). The output for each data collection should be a single integer representing the minimum number of tables needed to satisfy all of the group requests.

The output is to be formatted exactly like that for the sample output given below.

Assumptions: The value of *n* will be between 1 and 1000, inclusive.
Each input integer will be between 1 and 8, inclusive.
Each data collection will contain between 1 and 40 integers, inclusive.

Sample Input:

```
3
5 5 5
5 1 1 1 8
4 3 4 3 2
```

Sample Output:

```
3
2
2
```

Problem 9.5 **TransDate** (page 1 of 2)

General Statement: Dates are expressed in a wide variety of formats. A common format is `mm/dd/yyyy` which means expressing the month in two digits followed by a slash, the day of the month in two digits, another slash, and the year in four digits. The date `April 29, 2007` would be expressed as `04/29/2007` in this format. A similar format, `m/d/yy` only uses one digit for the month and one digit for the day of the month if possible. In addition, it only displays the last two digits of the year. Thus, the same date as above would be expressed as `4/29/07` in this format.

The individual formats for the month, day, and year can be rearranged in any order. For example, another popular date format is `yyyy/mm/dd` which would represent the above date as `2007/04/29`. The following list summarizes the formats of the date elements:

`mm`	Express month in two digits
`m`	Express month in one digit (if possible)
`dd`	Express day of month in two digits
`d`	Express day of month in one digit (if possible)
`yyyy`	Express year in four digits
`yy`	Express year using last two digits

Your task is to write a program that displays a date in a specific format given the same date expressed in a different format using the above components.

Input: The first line of the input contains an integer *n* that represents the number of data collections that follow where each data collection is on a single line. Each line contains three values *x*, *y*, and *z* with a single space separating each pair of values. The first value, *x*, is a date format using a combination of three of the elements in the above list. This represents the "old" date format. The second value, *y*, is a date given in that format. The final value, *z*, is a different date format also using a combination of three of the above elements. This represents the "new" date format.

Name of Data File: pr95.dat

Output: Your program should produce *n* lines of output (one for each data collection). The output for each data collection should be the input date displayed using the "new" date format. Slashes (`/`) should be used to separate each component.

The output is to be formatted exactly like that for the sample output given below.

Problem 9.5 TransDate (page 2 of 2)

Assumptions: The value of *n* will be between 1 and 100, inclusive.
The separators of the date elements will always be slashes (`/`).
The input date will always be valid.
An input date given with a two digit year should be assumed to be prefixed with “`20`” (occurs between 2000 and 2099, inclusive). Thus, `04/29/07` should be interpreted as `April 29, 2007`.

Sample Input:

```
3
m/d/yyyy 4/1/2010 yy/mm/dd
d/m/yy 9/12/24 mm/dd/yyyy
yyyy/mm/dd 1999/10/04 yy/d/m
```

Sample Output:

```
10/04/01
12/09/2024
99/4/10
```

Problem 9.6 An Honor Just to be Nominated…

General Statement: It's award season, and an organization needs to determine the nominees for each of its awards by counting the number of votes submitted. When a vote is cast, the voter simply writes the ID letter of its choice and sends it to the organization's governing body. Thus, the collection of votes can be represented as one long string of ID letters. When all of the votes are cast, the organization uses the *k* choices that received the most votes as its sets of nominees where *k* is some integer. Given a string of votes and the number *k*, your task is to identify the nominees.

Input: The first line of the input contains an integer *n* that represents the number of data collections that follow where each data collection is on a single line. Each data collection contains two components separated by a single space. The first component is a string of capital letters where each letter represents a single vote. The second component is an integer, *k*, representing the number of desired nominees.

Name of Data File: pr96.dat

Output: Your program should produce *n* lines of output (one for each data collection). The output for each data collection should be a string containing *k* capital letters with each letter representing a different nominee. The letters should be given in alphabetic order (not in the order of the number of votes).

The output is to be formatted exactly like that for the sample output given below.

Assumptions: The value of *n* will be between 1 and 1000, inclusive.
The value of each *k* will be between 1 and 26, inclusive.
The length of each vote string will be between 1 and 100, inclusive.
There will not be a tie for the nominee receiving the kth number of votes.
Each nominee list will have at least *k* different IDs.

Sample Input:

```
3
ABCDEABCDEABCD 4
XYYZZZWWWW 1
XAXBXCXDXAX 2
```

Sample Output:

```
ABCD
W
AX
```