Computer Science Competition
Invitational B 2024
Programming Problem Set

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.

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.

5. A penalty of 5 points will be assessed each time that an incorrect solution is submitted. This penalty will only be assessed if a solution is ultimately judged as correct.

II. Names of Problems

Number	Name
Problem 1	Ajeet
Problem 2	Billy
Problem 3	Daniella
Problem 4	Donghai
Problem 5	Hiromi
Problem 6	Jana
Problem 7	Ksenia
Problem 8	Marta
Problem 9	Prateek
Problem 10	Sofia
Problem 11	Tyler
Problem 12	Vika

1. Ajeet

Program Name: Ajeet.java **Input File: None**

Ajeet is making travel arrangements for his Leaguetown HS CompSci Team to go to Austin for the State UIL Meet in May 2024. To do this, he needs a calendar of May 2024.

Your job is to display the calendar for him. It should look exactly as shown below.

Input: None

Output: Duplicate the calendar exactly as shown below. Make sure to include the line below the month consisting of the six sets of digits 0-9.

Sample input: None

Sample output:

```
                          MAY 2024
01234567890123456789012345678901234567890123456789012345678
    SUN      MON      TUE      WED      THR      FRI      SAT
                                1        2        3        4
     5        6        7        8        9       10       11
    12       13       14       15       16       17       18
    19       20       21       22       23       24       25
    26       27       28       29       30       31
```

2. Billy

Program Name: Billy.java **Input File: billy.dat**

There is a lot of controversy this year because of the selections made to the Pickle Ball play-offs. There was so much discussion as to which teams made the Top Four. We will be talking about this for a long time.

Billy has been charged with finding a program that will find the winner among the eight schools in the league. He is given their names and their win-loss records. He must find the school with the highest winning percentage.

Now of course, due to a recent court decision, there will always be a clear winner after all the games are played. There is a guarantee not to have a tie for first place. So, it is all based on which team has the best winning percentage, even though teams possibly played a different number of games.

In the sample, Texas has the highest winning percentage at 90.9% with a record of 10 wins and 1 loss.

Input: There will be 16 lines of data. The odd numbered lines will be the team names. Each team name will be of length in the range [1,20]. Some teams may have spaces in their names. On the even numbered lines, there will be two integers, both in the range [0,24]. There will be one space between separating the numbers that are on each line. The first number is the number of wins. The second number is the number of losses. The win-loss record will always be on the line below the team name.

Output: Output the name of the team with the highest winning percentage.

Sample input:

```
MICHIGAN
12 2
WASHINGTON
7 14
TEXAS
10 1
ALABAMA
4 9
FLORIDA STATE
10 2
GEORGIA
1 12
OHIO STATE
4 5
OREGON
7 10
```

Sample output:

```
TEXAS
```

3. Daniella

Program Name: Daniella.java **Input File: daniella.dat**

Daniella is concerned with words. She is always thinking about words. She has always been that way. She understands language and loves to think about words.

In her job as a chef, she is in charge of preparing meals for the customers in a fancy restaurant. In that job, she cooks great meals and is in charge of every step in the process. One of those steps is chopping up the vegetables. She is known world-wide for her chopping skills.

Every once in a while, her mind wanders, and in her daydreams, she combines her two interests. Sometimes even, she imagines chopping words. Your job is to write a program to "chop" words into pieces.

You will be given a String S and an integer N. Your job is to chop S into groups containing N letters each. Between each word piece, you will place a dash, but you will not have a dash at the beginning or the end of the word. As you move across the word from left to right, place dashes after each group of N letters. The last group may be shorter than the rest if there are not enough letters to create a group of N.

Examples:

`COMPUTER 2` produces `CO-MP-UT-ER`
`GLASS 1` produces `G-L-A-S-S`
`PROGRAM 3` produces `PRO-GRA-M`
`FROG 5` produces `FROG`

Input: The first line of data will contain an integer T representing the number of data lines to follow. T will be in the range [1,20]. The next T lines of data will contain a string and an integer with one space in between. The string will have a length in the range [1,40]. It will consist of uppercase letters. The integer will be an integer with a value in the range [1,5].

Output: Each output will consist of uppercase letters separated by dashes if appropriate.

Sample input:

```
6
RIGATONI 2
PENNE 5
FETTUCCINE 4
LASAGNA 1
MACARONI 5
SPAGHETTI 3
```

Sample output:

```
RI-GA-TO-NI
PENNE
FETT-UCCI-NE
L-A-S-A-G-N-A
MACAR-ONI
SPA-GHE-TTI
```

4. Donghai

Program Name: Donghai.java **Input File: donghai.dat**

Donghai helped Joan solve the Invitational A problem counting words and calculating the average size of words. Now, he wants to work with letters instead of words. He wants to count the number of times each letter occurs in written prose. He will ignore punctuation marks and any other non-alphabetic characters. Uppercase and lowercase letters are considered the same letter. Once they have been counted, he will display the list of letters that occurred.

Can you help Donghai solve this problem?

Input: First line will contain an integer N with $0 < N \leq 10$ which is the number of paragraphs that follow. Each paragraph will occur on a single line of text and will not exceed 2000 characters in total length. Any ASCII character can occur in a paragraph but the only ones of interest are the standard alphabetic letters, both uppercase A…Z and lowercase a…z.

Output: Each test case must display list of letters in uppercase that have non-zero counts. Each output line with a letter will start with that letter followed by a colon "**:**" followed by the count with no spaces. Following each test case display a line containing 10 equal signs "==========".

Sample input: (*indented lines are continuation of previous line*)

```
3
Let's start with a short 1-line paragraph just to see a result!
Now, this second paragraph will be much longer which results in the sample input
   displaying with all continuation lines indented.  However, it will be a single
   unbroken line in the data file that is provided for the contest...
This is the third "paragraph" of sample data for this programming problem
```

Sample output:

```
A:6
E:5
G:1
H:3
I:2
J:1
L:3
N:1
O:2
P:2
R:5
S:6
T:8
U:2
W:1
==========
```

Sample output: *continued*

```
A:11
B:3
C:5
D:7
E:20
F:2
G:4
H:11
I:20
K:1
L:14
M:2
N:18
O:10
P:6
R:8
S:10
T:16
U:5
V:2
W:6
Y:1
==========
```

Sample output: *continued*

```
A:7
B:1
D:2
E:3
F:2
G:3
H:5
I:5
L:2
M:4
N:1
O:4
P:5
R:7
S:4
T:5
==========
```

5. Hiromi

Program Name: Hiromi.java **Input File: hiromi.dat**

Sudoku is a popular, logic-based game that anyone, who can count from 1-9, can play! A valid sudoku puzzle is a 9 by 9 grid of numbers such that each row, each column, and each 3x3 sub-grid region has the numbers 1-9 with no missing numbers or no duplicate numbers. A puzzle is traditionally given incomplete, and it is up to the player to fill in all the missing spaces. As the below puzzle shows, each row (horizontal), each column (vertical), and each of the nine 3x3 sub-grids, all have the numbers 1-9. See the below puzzle for a valid Sudoku puzzle, as well as the layout of the 9 sub-grids. (The darker, bolder lines denote the sub-grids.)

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Hiromi loves solving Sudoku's but is known to make a mistake time-to-time. In an effort to ensure the final attempts are correct, Hiromi would like your help writing source code that verifies if a given attempt at a Sudoku puzzle is in fact correct, or not. Can you help with this?

Input: Input will begin with an integer N, the number of test cases. N is in the range [1,10]. Each of the following N test cases will start with 17 dashes to serve as a divider between each of the puzzles. Following the dashes will be the 9 x 9 Sudoku puzzle. All, individual numbers will be separated by a space to simplify the reading of the input. Each number present is guaranteed to be in range of [1,9].

Output: For each test case, you are to output: "`Puzzle #X: true`" if the puzzle meets the described criteria for a valid Sudoku puzzle, or "`Puzzle #X: false`", where `X` is the current test case. Remember, all nine rows, all nine columns, and all 9 sub-grids must have the numbers 1-9, to be a valid puzzle.

Sample input and output on next page…

~ Hiromi continued… ~

Sample input:

```
5
-----------------
5 3 4 6 7 8 9 1 2
6 7 2 1 9 5 3 4 8
1 9 8 3 4 2 5 6 7
8 5 9 7 6 1 4 2 3
4 2 6 8 5 3 7 9 1
7 1 3 9 2 4 8 5 6
9 6 1 5 3 7 2 8 4
2 8 7 4 1 9 6 3 5
3 4 5 2 8 6 1 7 9
-----------------
1 2 3 4 5 6 7 8 9
4 5 6 7 8 9 1 2 3
7 8 9 1 2 3 4 5 6
2 3 4 5 6 7 8 9 1
5 6 7 8 9 1 2 3 4
8 9 1 2 3 4 5 6 7
3 4 5 6 7 8 9 1 2
6 7 8 9 1 2 3 4 5
9 1 2 3 4 5 6 7 8
-----------------
5 3 4 6 7 8 9 1 2
6 7 2 1 9 5 3 4 8
1 9 8 3 4 2 5 6 7
8 5 9 7 6 1 4 2 3
4 2 6 8 5 3 7 9 1
7 1 3 9 2 4 8 5 6
9 6 1 5 3 7 2 8 4
2 8 7 4 1 9 6 3 5
3 4 5 2 8 6 1 7 7
-----------------
8 3 5 4 1 6 9 2 7
2 9 6 8 5 7 4 3 1
4 1 7 2 9 3 6 5 8
5 6 9 1 3 4 7 8 2
1 2 3 6 7 8 5 4 9
7 4 8 5 2 9 1 6 3
6 5 2 7 8 1 3 9 4
9 8 1 3 4 5 2 7 6
3 7 4 9 6 2 8 1 5
-----------------
5 3 4 6 7 8 9 1 2
6 7 2 1 9 5 3 4 8
1 9 8 3 4 2 5 6 7
8 5 9 7 6 1 4 2 3
4 2 6 8 5 3 7 9 1
7 1 3 9 2 4 8 5 6
9 6 1 5 3 7 2 8 4
2 8 7 4 1 9 6 3 5
3 4 5 2 8 6 1 7 3
```

Sample output:

```
Puzzle #1: true
Puzzle #2: true
Puzzle #3: false
Puzzle #4: true
Puzzle #5: false
```

6. Jana

Program Name: Jana.java **Input File: jana.dat**

Jana is opening a baseball training facility and needs your help making some targets to help the pitchers practice their accuracy. You will be given the size of the target, and you need to print out the front of it to be painted onto the target. Each target will be size `ixi`, with rings of `'#'` characters, moving inwards and having every other ring be made of spaces (see sample output).

Input:
The input will begin with an integer `n`, denoting the number of test cases to follow. Each test case will consist of an integer, `i`, on its own line, denoting the height and width of the target to be printed.

Output:
Output the `ixi` target design as described above and shown in the sample output.

Sample input:
```
3
6
9
7
```

Sample output:
```
######
#    #
# ## #
# ## #
#    #
######
#########
#       #
# ##### #
# #   # #
# # # # #
# #   # #
# ##### #
#       #
#########
#######
#     #
# ### #
# # # #
# ### #
#     #
#######
```

7. Ksenia

Program Name: Ksenia.java **Input File: ksenia.dat**

Ksenia is very interested in graphs, especially what happens when you add a copy of a graph to another one. When Ksenia makes a copy of a graph and adds it to the original copy, she performs the following: for each vertex in the first copy, add an edge from that vertex to the corresponding vertex in the second copy.

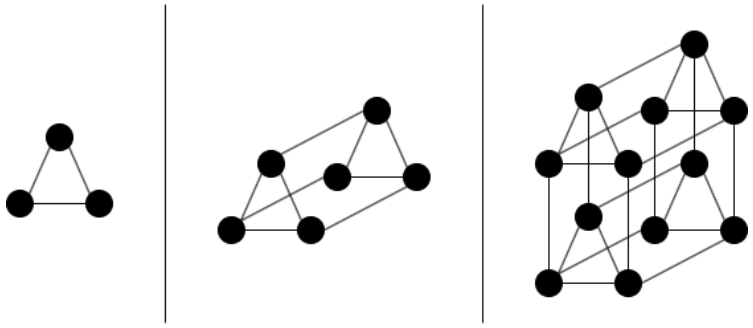

Figure 1 - Visualization of the first two copies of the first graph from the sample input

While conceptualizing what some of these graphs might look like might become difficult for graphs that have been copied multiple times, Ksenia simply is curious in the number of edges and vertices that a given graph will have once it has been copied a particular number of times. Help Ksenia by writing a program that, given a graph and the number of times that Ksenia wishes to duplicate that graph, returns the number of vertices and edges in that updated graph.

Input: The first line of input will consist of an integer n, denoting the number of queries to follow. For each of the n queries, the first line of input will consist of two integers, V and E, denoting the number of vertices and edges in the graph, where vertices are numbered 0 through $V-1$. Thet next line will consist of E comma-separated pairs of space-separated integers v_a and v_b denoting two vertices that form an edge in the graph. The final line will consist of a single integer m, denoting the number of times that Ksenia wishes to duplicate the graph.

It should be noted that $n \geq 1$, $V \geq 1$, $E \leq \frac{V(V-1)}{2}$, and $m \geq 1$. For a given query, if $E = 0$, note that there won't be a line denoting the edges. Lastly, each of the E edges will be unique, and there will never exist an edge between a given vertex and itself.

Output: For each of the n queries, print on its own line two space-separated integers V' and E' denoting the number of vertices and edges, respectively, in the final copy of the graph after it has been duplicated m times.

Sample input:

```
3
3 3
0 1,1 2,2 0
3
1 0
7
5 4
0 1,1 2,2 0,3 4
4
```

Sample output:

```
24 60
128 448
80 224
```

8. Marta

Program Name: Marta.java **Input File: marta.dat**

Marta has been studying hard for the upcoming UIL season but is having some difficulty with the classic bubble sort for arrays. She understands that every pass through the array will move smaller items one direction and larger items the other direction. An ascending sort will move smaller items towards the lower or [0] array position while larger items move towards the higher-end of the array. A descending sort would move items the opposite directions.

She would like to be able to confirm her manual sort progress after a specific number of passes to view how the arrangement should be changing. Here is an example:

Initial	95	11	68	13	33	26	24	60	56	47	79	21
Pass 1	11	68	13	33	26	24	60	56	47	79	21	95
Pass 2	11	13	33	26	24	60	56	47	68	21	79	95
Pass 11	11	13	21	24	26	33	47	56	60	68	79	95

However, the sort may not get to the requested pass count, depending on how quickly the numbers settle into their correct positions. In that case, the numbers will be fully sorted but Marta wants to know how many passes it actually took to complete the sort when that happens.

Can you help Marta modify her sort to output a list of numbers a specific pass of the sort?

Input: First line will contain an integer T with $1 \leq T \leq 10$, the number of test cases. Each test case will consist of two lines of space-separated integer data. First line will contain an integer N with $10 \leq N \leq 50$, the number of integers to be sorted, followed by another integer P with $1 \leq P < N$, the sort pass after which the array content will be displayed. The next line will contain N integers in range [10, 99], the data to be sorted into ascending order but displayed after only P or fewer passes have been completed.

Output: Each test case will produce 1 line of output that starts with "`Test case #1 pass #2: `" where `#1` is the test case number and `#2` is the actual number of passes completed which can be less than P because of the early exit technique from a proper bubble sort.

Sample input:

```
3
12 1
95 11 68 13 33 26 24 60 56 47 79 21
12 2
95 11 68 13 33 26 24 60 56 47 79 21
25 23
71 40 52 66 95 80 61 56 26 64 34 47 99 51 96 86 63 55 49 72 82 43 93 42 76
```

Sample output: (*lines that are indented are continuation of previous line*)

```
Test case 1 pass 1: 11 68 13 33 26 24 60 56 47 79 21 95
Test case 2 pass 2: 11 13 33 26 24 60 56 47 68 21 79 95
Test case 3 pass 21: 26 34 40 42 43 47 49 51 52 55 56 61 63 64 66 71 72 76 80 82 86 93
   95 96 99
```

9. Prateek

Program Name: Prateek.java **Input File: prateek.dat**

Prateek is a fan of the binary search algorithm and is looking at the efficiency of the algorithm. In particular, he is curious in the number of steps that it takes to find a given element in a sorted list using the binary search algorithm. A copy of the binary search algorithm has been provided to the right.

```
BINARYSEARCH(A, s, t)
  let A be the list of numbers
  let s be the size of the list
  let t be the target
  l ← 0
  r ← s − 1
  m ← ⌊(l+r)/2⌋
  while (l ≤ r)
    if (A[m] == t)
      return m
    else if (A[m] < t)
      r ← m − 1
    else
      l ← m + 1
    m ← ⌊(l+r)/2⌋
```

As a result, Prateek wants to classify each item in a sorted list by how quickly each item will be found using the algorithm. More specifically, for each item in the list, Prateek wants to put that element in a set, where each element in a given set share the same number of iterations of the binary search algorithm's while loop needed to be discovered.

Help Prateek solve this problem by writing a program that breaks down a sorted list into unique sorted sets where each element in a given set has the same number of iterations needed as the other elements in the set.

Input:
The first line will consist of a single integer, n, denoting the number of lines to follow. The next n lines will each consist of a space-separated list of sorted integers. It should be noted that $1 \leq n \leq 50$, and the length of any given list will be no more than 100.

Output: For each of the n sorted lists, output the analysis of the list as follows: on the first line, print the string "`List: <LIST>`", where `<LIST>` is a space-separated string representation of the sorted list. Then, for each unique set generated by Prateek's program, print out "i`-iterations: <SET>`", where i is the number of iterations needed to find the elements contained in `<SET>`, and `<SET>` is a space-separated string representation of the sorted set of elements which all take i iterations of the binary search algorithm to find. Print out the sets in increasing order in relation to the value of i.

Sample input:
```
3
2 4 6 8 9
1000
2 3 5 7 11 13 17 19 23 27 29 31
```

Sample output:
```
List: 2 4 6 8 9
1-iterations: 6
2-iterations: 2 8
3-iterations: 4 9

List: 1000
1-iterations: 1000

List: 2 3 5 7 11 13 17 19 23 27 29 31
1-iterations: 13
2-iterations: 5 23
3-iterations: 2 7 17 29
4-iterations: 3 11 19 27 31
```

10. Sofia

Program Name: Sofia.java **Input File: sofia.dat**

Sofia (who you met while on vacation with your friend Rodrigo), is going on vacation, and needs to find out if she can get from certain cities to certain other cities with the amount of money she has saved for her traveling. You will be given the prices to fly between two cities, and then two cities and an amount of money. You need to write a program to determine if that is enough money to fly between the two given cities. You need to determine the shortest path from the starting to the ending city, and find out if the amount of money Sofia has will be enough to get between the two cities.

Input: The input will begin with two integers, f, denoting the number of flights between 2 cities listed, and n, denoting the number of test cases. The next f lines will each contain 2 strings and an integer, all separated by spaces, denoting the names of the two cities and the price of the flight in either direction, respectively. The following n lines will each contain a test case, containing two strings and an integer, denoting the starting city and ending city, and the amount of money Sofia has to get from the starting to the ending city.

Output: For each test case, if there is enough money to get from the starting city to the ending city output the string "`[The name of the ending city] is always a good idea.`". Otherwise, output "`There's no place like home.`".

Sample input:

```
6 2
Paris Dallas 1200
Paris Prague 650
Prague Munich 300
Paris Munich 300
Berlin Prague 400
Berlin Dallas 2300
Dallas Prague 1550
Dallas Berlin 2200
```

Sample output:

```
There's no place like home.
Berlin is always a good idea.
```

11. Tyler

Program Name: Tyler.java **Input File: tyler.dat**

Tyler is a rich philanthropist and is a big fan of the trend of "doubling it and giving it to someone else." This trend consists of starting with a single dollar and giving participants the option of either doubling the current dollar amount and giving it to someone else, or keeping the current amount of money for themselves. This continues until someone chooses to take the money for themselves. Due to Tyler's large wealth, he has larger ambitions.

Rather than simply doubling the current amount of money and giving it to someone else, Tyler prefers giving his participants options of what to multiply the current amount of money by. As a result, he is curious whether certain dollar amounts are obtainable given a list of options to multiply the current amount of money by. You have been contracted by Tyler to develop a program which can help him automatically solve such a problem.

Input: The first line of input will consist of two integers, n and m – the number of options for multipliers and the number of queries, respectively. The next line will consist of a list of n space-separated integer multipliers. The following line will consist of a list of m space-separated integer queries indicating target dollar amounts that Tyler is interested in. It should be noted that $1 \leq n, m \leq 10^3$.

Output: For each of the m queries, print on its own line the string "Target <t>: <ANS>", where <t> is the ith target and <ANS> is the string "YES" or "NO" corresponding to whether or not the ith target is obtainable using any sequence of the m multipliers, each of which can be used any number of times.

Sample input:

```
5 10
2 3 5 7 11
2 13 5 26 121 77 15 144 154 5336100
```

Sample output:

```
Target 2: YES
Target 13: NO
Target 5: YES
Target 26: NO
Target 121: YES
Target 77: YES
Target 15: YES
Target 144: YES
Target 154: YES
Target 5336100: YES
```

12. Vika

Program Name: Vika.java **Input File: vika.dat**

You and your new friend in English class, Vika, have been playing a game. Vika gives you 2 words and its your job to find the longest sequence of letters that exists in both words. This sequence can be made up of any characters in the words, the only condition is that they appear in the same order in both words. For example, the sequence AC is common to both BAC and ABC.

Input: The input will begin with an integer, `n` (`0 < n <= 1000`), denoting the number of test cases to follow. Each test case will consist of two space separated strings for you to play the game with. Determine the length of the longest common subsequence of these two strings.

Output: Output the length of the longest possible subsequence that is shared between these two strings.

Sample input:

```
3
ABC BAC
Someone Stonecold
Applebees Geronimo
```

Sample output:

```
2
4
1
```