Computer Science Competition District 2025

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	Atlas
Problem 2	Becker
Problem 3	Brian
Problem 4	Chalam
Problem 5	Dean
Problem 6	Dominick
Problem 7	Dorian
Problem 8	Erbatur
Problem 9	Jacinta
Problem 10	Kassie
Problem 11	Luke
Problem 12	Sage

1. Atlas

Program Name: Atlas.java **Input File: none**

Atlas is excited to be competing on his UIL team. He can't control his excitement and has to have a banner to express it for all to see.

Input: None

Output: The banner exactly as shown below.

Sample input: None

Sample output:
```
:'######:::'######:::::'########:::'#######:::'######::'##:::'##::'######::'####:
'##... ##:'##... ##:::: ##.... ##:'##.... ##:'##... ##: ##::'##::'##... ##: ####:
 ##:::..:: ##:::..::::: ##:::: ##: ##:::: ##: ##:::..:: ##:'##::: ##:::..:: ####:
 ##:::::::. ######::::: ########:: ##:::: ##: ##::::::: #####::::. ######::: ##::
 ##::::::::..... ##:::: ##.. ##::: ##:::: ##: ##::::::: ##. ##::::..... ##::..:::
 ##::: ##:'##::: ##:::: ##::. ##:: ##:::: ##: ##::: ##: ##:. ##::'##::: ##:'####:
. ######::. ######::::: ##:::. ##:. #######::. ######:: ##::. ##:. ######:: ####:
:......::::......::::::..:::::..:::.......::::......:::..::::..:::......:::....::
```

2. Becker

Program Name: Becker.java **Input File: becker.dat**

Becker, a friend of yours, has become a finance guru. They're talking to you about investing and compound interest, so you get curious. You ask them how long it would take you to reach a certain amount of money if you were to invest. Becker replies that it depends on several factors and provides you with this nifty formula below. You, being a programming genius, decide to take it upon yourself to write a program to calculate this value.

A - Amount goal
P - Principal (initial investment)
r - interest rate as a decimal
n - number of times compounded per year
t - time invested (in years)

$$A = P\left(1 + \frac{r}{n}\right)^{nt}$$

Input: Each input line will contain 4 space-separated values A, r, n, t. Where (1 <= A <= 10^18), (1 <= r <= 100), (1 <= n <= 52), and (1<= t <= 100). All values will be whole numbers (r is a percent).

Output: The minimum whole-dollar rounded principal that needs to be invested to reach the goal in the specified number of years.

Sample input:
```
1000 4 12 1
766777 10 12 30
```

Sample output:
```
961
38654
```

3. Brian

Program Name: Brian.java **Input File: brian.dat**

Your neighbor, Brian, is a milk delivery man. He drives a rather peculiar route though. Some of the streets he drives down take him back in time. Because of this, he makes sure to never take the same street twice when running his route. He has a rather important client he must deliver milk to next week, and he has been stressing trying to figure out what the shortest route to him is given this complication. He remembers you talking about finding shortest paths recently in your Computer Science class, so he has asked you to help him with this task.

Input: The first input line will contain a single number N (1 <= N <= 100) that denotes the number of test cases that follow. Each test case will start with two space-separated integers D and S (1 <= D, S <= 10^4), being the number of milk drop-off locations, and the number of streets connecting these locations. The following S lines will contain three space-separated values: the names of two locations followed by an integer value denoting the number of minutes it takes to travel between them. The final line of each test case will contain two space-separated values Start and Stop denoting the name of the starting drop-off and the name of the finishing drop-off respectively.

Output: The smallest amount of time it would take Brian to reach the stop location from the start location. If the amount of time is nondeterministic, print out "Take as long as you need.".

Sample input:
```
1
10 14
Moo-Manor Pasteur-Palace 10
Pasteur-Palace Udderly-Central 3
Udderly-Central The-Dairy-Queen -4
Udderly-Central The-Milky-Way 21
Udderly-Central Moo-ve-In-Ready -12
The-Dairy-Queen Latte-Lodge 20
The-Dairy-Queen Lunchbox-Lair 18
The-Milky-Way Casa-del-Cream -9
Udderly-Central Ouch-House 27
Pasteur-Palace Ouch-House -17
Casa-del-Cream Moo-Manor 22
Moo-Manor Lunchbox-Lair -5
LunchBox-Lair Latte-Lodge 19
Latte-Lodge Pasteur-Palace -13
The-Milky-Way Ouch-House
```

Sample output:
```
6
```

4. Chalam

Program Name: Chalam.java **Input File: chalam.dat**

Your friend Chalam has taken a recent fascination in the fundamental theorem of arithmetic which states that every positive integer can be represented uniquely as a product of prime numbers. Chalam's fascination has been in finding that unique prime representation for different numbers. However, he has recently come to a standstill as the numbers he wishes to break down are becoming increasingly larger and larger, making the amount of time that it takes to identify the unique classification unbearably long. Knowing a thing or two about how to automate this process, you decide to write Chalam a program which, given a positive integer, determines the unique prime product representation of that number.

Input: The first line of input will consist of a single integer, n ($1 \leq n \leq 2.5 \cdot 10^4$), denoting the number of integers that need to be processed. The next n lines will each consist of a single integer

q_i, (for all i, $1 \leq i \leq n$: $2 \leq q_i \leq 2^{31} - 1$),

the i^{th} of which represents Chalam's i^{th} query.

Output: For each of Chalam's n queries, each on their own line, output a space-separated list of the prime numbers, which, when multiplied with one another, equal Chalam's query. In cases where a given prime number is listed more than once, condense the expression into the form of ${p_j}^{e_j}$ (`p_j^e_j`) where p_j represents the j^{th} prime number and e_j represents the number of times that p_j occurs in the product. Lastly, these terms should be listed in ascending order according to the value of the base of the exponent (i.e., terms should be listed such that $p_1 < p_2 < \cdots < p_j$).

Sample input:
```
20
2
3
4
10
11
12
13
20
21
22
23
24
30
40
51
67
91
103
108
245
```

Sample output:
```
2
3
2^2
2 5
11
2^2 3
13
2^2 5
3 7
2 11
23
2^3 3
2 3 5
2^3 5
3 17
67
7 13
103
2^2 3^3
5 7^2
```

5. Dean

Program Name: Dean.java **Input File: dean.dat**

Your friend Dean has recently learned about Palindromes in his CS1 class. As he explains to you, a Palindrome is any string that is the same forwards and backward. Naturally, this includes a single character. Dean has been going through random books and picking out sentences, smashing them together, and counting the number of substrings that are palindromes. A substring, as he also explains to you, is any continuous run of characters in a string.

Input: The first input line will contain a number N (1 <= N <= 10) defining the number of strings to check. The following N lines will contain a single string. The length of each string will be a whole number M such that 1 <= M <= 10^5.

Output: Output the number of substrings that are palindromic for each string.

Sample input:
```
3
abc
aaa
213
```

Sample output:
```
3
6
3
```

6. Dominick

Program Name: Dominick.java **Input File: dominick.dat**

You and Dominick were walking in the park when you suddenly came across a stack of Snails. The snails were stacked in order of decreasing size, with the largest on the bottom and the smallest on top. A cool rock is nearby, and Dominick decides he wants the snails stacked on the rock. But he's afraid that picking them all up at once might be dangerous, so he wants to move them one at a time. So, he gets a second rock and starts moving them over one at a time, being careful to never stack a bigger snail on top of a smaller snail. Write a program that, given a stack of snails, shows the order in which Dominick would have to move the snails to move the entire stack to the cool rock.

Input: The first input line will contain a value N (1 <= N <= 100) denoting the number of input lines that follow. The remaining input lines will contain an unknown number of integer values V (1 <= V <= 10) representing the size of the snails, in order from bottom to top. You can assume that no two snails will be the same size, and the snails will be given in descending order.

Output: Output the snail that Dominick moves at each step along the way of moving the stack to the rock. The output should be in the format "Move snail S from rock X to rock Y" where the rocks are numbered 1,2,3 and S is the position of the snail from the top of the starting pile (1 is the top). 1 is the starting rock, and 3 is the final rock. There should be a blank line separating each tower of snail's output.

Sample input:

```
2
3 2 1
1
```

Sample output:

```
Move snail 1 from rock 1 to rock 3
Move snail 2 from rock 1 to rock 2
Move snail 1 from rock 3 to rock 2
Move snail 3 from rock 1 to rock 3
Move snail 1 from rock 2 to rock 1
Move snail 2 from rock 2 to rock 3
Move snail 1 from rock 1 to rock 3

Move snail 1 from rock 1 to rock 3
```

7. Dorian

Program Name: Dorian.java Input File: dorian.dat

Dorian is putting the final touches on the problems that he wrote for his school's upcoming Competitive Programming invitational and now needs to write the Judge Documents for his problems. However, Dorian finds himself in a bit of a pickle since his judge data is really, really long and doesn't want to format it by hand when copying it over from the raw files into the Word/PDF documents. Namely, when a line of input or output goes beyond 89 characters, that line will need to be broken down into multiple lines, each additional line beyond the first needing to be indented with a "tab" (which you should treat as 6 spaces) to show continuity. However, he doesn't want to willy-nilly break down a long line – namely, he never wants to break a contiguous set of non-whitespace characters up, even if it means having an excessive number of continued lines, thus wasting a fair bit of otherwise usable space. Knowing that a set of contiguous non-whitespace characters that go beyond 89 characters long (even 83 for indented lines) pose a challenge to his desired format, Dorian has specially crafted his problems to ensure that this never happens. Dorian would normally write this program himself, but he is quite tired after writing 12 problems for his contest. Help Dorian by writing a program that, given a raw data file, formats it in Dorian's desired format.

Input: The input will consist of an unknown number of lines, each line of input will consist of, at minimum, one non-whitespace character. The first line of input that is empty or does not consist of any non-whitespace character will denote the end of the input.

Output: For each line of input, output a new (potentially more than one) line(s) of output that is formatted per Dorian's request. Note that internal whitespace within a line of input should be respected in the output unless doing so would either cause Dorian's rule about contiguous non-whitespace characters to be violated, or if it were to cause the first non-tab character on a "continued line" to be a non-whitespace character. In such cases, the original whitespace should be completely ignored and disregarded in the output.

Sample input: (***indented lines are a continuation of the previous line; lines made longer to reflect width of judge document***)

```
12345678901234567890123456789012345678901234567890123456789012345678901234567890123456789
      1234567890 1234567890  1234567890   1234567890    1234567890     1234567890
      1234567890       1234567890        1234567890         1234567890
ABCDEFGHIJKLMNOPQRSTUVWYZ    ABCDEFGHIJKLMNOPQRSTUVWYZ  ABCDEFGHIJKLMNOPQRSTUVWYZ
      ABCDEFGHIJKLMNOPQRSTUVWYZ   ABCDEFGHIJKLMNOPQRSTUVWYZ  ABCDEFGHIJKLMNOPQRSTUVWYZ
The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog.
      The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the
      lazy dog. The quick brown fox jumps over the lazy dog. The quick brown fox jumps
      over the lazy dog. The quick brown fox jumps over the lazy dog. The quick brown fox
      jumps over the lazy dog. The quick brown fox jumps over the lazy dog.
```

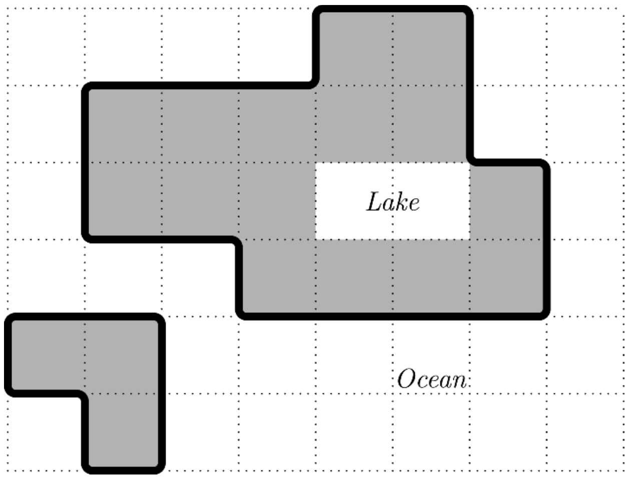
Sample output: (***indented lines are <u>no longer</u> a continuation of the previous line*** 😉)

```
12345678901234567890123456789012345678901234567890123456789012345678901234567890123456789
      1234567890 1234567890  1234567890   1234567890    1234567890     1234567890
      1234567890       1234567890        1234567890         1234567890
ABCDEFGHIJKLMNOPQRSTUVWYZ    ABCDEFGHIJKLMNOPQRSTUVWYZ  ABCDEFGHIJKLMNOPQRSTUVWYZ
      ABCDEFGHIJKLMNOPQRSTUVWYZ   ABCDEFGHIJKLMNOPQRSTUVWYZ  ABCDEFGHIJKLMNOPQRSTUVWYZ
The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog.
      The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the
      lazy dog. The quick brown fox jumps over the lazy dog. The quick brown fox jumps
      over the lazy dog. The quick brown fox jumps over the lazy dog. The quick brown fox
      jumps over the lazy dog. The quick brown fox jumps over the lazy dog.
```

8. Erbatur

Program Name: Erbatur.java **Input File: erbatur.dat**

Erbatur loves spending his summers in the Bahamas but has always struggled to pick which of the islands in the Bahamas to stay at. One factor that is important to Erbatur is how large of an island he is staying on. More specifically, the length of the coastline (where the ocean meets the land of the island) plays a large factor in how much Erbatur enjoys where he stays. Knowing that planning trips is a large undertaking, rather than look at the entire map of the Bahamas, Erbatur decides to look at only a subsection of the map and wants to know how long the total coastline is of all islands contained within his view. Help Erbatur write a program that automates this process.



Gray squares represent land, solid black lines outline the coastline of two disjoint islands.

Input: The first line of input will consist of a single integer, T ($1 \leq T \leq 10^2$), denoting the number of test cases to follow. Each test case will begin with a single line of two space-separated integers n and m ($1 \leq n, m \leq 10^3$) denoting the size of the $n \times m$ map-view to follow. The next n lines will each consist of m characters, each of which is either a '.' to represent water, or a '#' to represent land. You may assume that the map-view of each test case is surrounded by ocean.

Output: For each of Erbaturs's T queries, each on their own line, output a single integer ℓ_i denoting the total length of the coastline of all islands, the ith of which corresponds to Erbatur's ith query. Note that water that appears within a given map view is not guaranteed to be a part of the ocean. For a cell of water to be considered a part of the ocean, it must be connected to the border of the map only through other cells of water.

Sample input:
```
5
6 8
....##..
.#####..
.###..#.
...####.
##......
.#......
5 6
.####.
.#.##.
###...
....#.
......
```
~ Input continues next column ~

~ Input continued ~
```
3 2
.#
..
#.
5 10
....#....#
...#....##
..#....##.
.#....##..
#....##...
3 5
.....
.....
.....
```

Sample output:
```
28
20
8
40
0
```

9. Jacinta

Program Name: Jacinta.java **Input File: jacinta.dat**

Jacinta has a problem. She's your boss so that means you have a problem. She needs a program ASAP, and your job is on the line. Your program needs to take an array of integers and determine if the array can be partitioned into two separate subarrays that sum to the same value. These subarrays do not need to be contiguous, subarrays can be formed any way you want, but each element can only be in one of the two subarrays.

Input: The input will begin with one integer, `n` (`0 < n <= 1000`), denoting the number of arrays to be checked. Each of the following n lines will contain an unspecified number of space-separated integers to be partitioned.

Output: For each test case, if the array can be partitioned into two parts that have equal sums, print the string "`Job secured.`". Otherwise, print "`Indeed here I come.`".

Sample input:
```
3
1 2 3 4 5 6
1 2 3 4 1 5 6
2 2 2 3 4 1 2
```

Sample output:
```
Indeed here I come.
Job secured.
Job secured.
```

10. Kassie

Program Name: Kassie.java **Input File: kassie.dat**

You and Kassie are working on a project together. You need to write a program that takes an expression in the following format, and solves for x:

```
(number1) (symbol) (number2) = x
```

`number1` and `number2` will be positive integer values, and `symbol` will be the symbol for either addition, subtraction, multiplication, or division. You simply need to determine what x would be. Note: all division will be integer division.

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 one line in the format listed above in the problem description.

Output: Output the integer value of x for the given expression.

Sample input:

```
4
23 + 4 = x
36 / 5 = x
212 - 17 = x
89 / 54 = x
```

Sample output:

```
27
7
195
1
```

11. Luke

Program Name: Luke.java **Input File: luke.dat**

Your friend Luke has begun working as a plumber. He needs you to write a program to determine the maximum water flow between two points in a given network of pipes, while respecting the maximum capacity of each pipe. Each pipe will connect between two “nodes” in the network, and will have a capacity associated with it denoting the maximum amount of water that can flow through the given pipe at a time.

Input: The input will begin with two space-separated integers, `n` (`0 < n <= 1000`) and `m` (`0 < m <= 1000`), denoting the number of pipes in the network, and the number of test cases to follow, respectively. Each of the following `n` lines will contain a pipe, with each pipe will be denoted by two space-separated characters, `a` and `b`, denoting the two nodes connected by this pipe (from `a` to `b`, not the other way around), followed by an integer denoting the capacity of the pipe. The next `m` lines will contain 2 characters, separated by a space, denoting the two nodes you need to find the maximum flow between. Note: the maximum flow will never exceed 100.

Output: For each test case, output the integer value denoting the maximum flow between the two given nodes.

Sample input:
```
10 2
A B 16
A C 13
B C 10
B D 12
C B 4
C E 14
D C 9
D F 20
E D 7
E F 4
A F
B E
```

Sample output:
```
23
14
```

12. Sage

Program Name: Sage.java **Input File: sage.dat**

Sage is working as a TA for her professor. She needs a program to deliver some statistics on exam grades.

Input: Each line in the file will consist of a five-digit ID number followed by an integer exam score in the range 0 <= score <= 100, separated by a single space. You will not be told how many lines of data are in the file.

Output: You need to output the average score to two decimal places, the highest score along with the ID of the student(s) with that score, the lowest score along with the ID of the student(s) with that score, and the number of students whose scores are above the average score. When displaying the data as shown below, each of the indented lines are each 3 spaces deeper on each level. The IDs need to be listed in the original order they were processed.

Sample input:
```
12345 85
67890 92
13579 78
24680 92
11223 65
33445 70
```

Sample output:
```
Average score: 80.33
   Highest score: 92
      Students with highest score:
         67890
         24680
   Lowest score: 65
      Students with lowest score:
         11223
Number of students above the average: 3
```