Rookie Code Rumble 2024 Problems

UNSW CPMSoc and UNSW CSESoc

24 June 2024

Contents

Rice Cooker

Time limit: 1 second | Memory limit: 1 gigabyte

CPMSoc just bought a new rice cooker and wanted to cook rice for their friends at CSESoc. However, before the event, Isaiah and Cyril couldn't agree on how much rice to cook beforehand.

They both forgot until the last minute and decided to bring as much rice and water as they could. That afternoon, when they met, Isaiah brought A cups of raw rice and Cyril brought B cups of water.

According to the instructions on the rice cooker, the raw rice to water ratio is $1 : 1$. If they cook as much rice as possible, how many cups of raw rice will they have left?

Input

You should read from standard input. We recommend using the templates at the top of the page to help you with input and output.

- The first and only line of input contains two integers A and B, where A is the initial number of cups of raw rice and B is the initial number of cups of water.

Output

You should write to standard output.

Output a single integer, the number of cups of raw rice are left after cooking as much rice as possible.

Constraints

For all test cases:

- $1 \leq A, B \leq 1000$.

Sample Input 1

```
10 5
```

Sample Output 1

```
5
```

Explanation 1

In this case, they can cook rice with 5 cups of raw rice and water, which leaves 5 cups of raw rice.

Sample Input 2

```
100 100
```

Sample Output 2

```
0
```

Explanation 2

In this case, both amounts are 100, so no raw rice will be left over.

Scoring

Your program will be run on the 2 sample cases and 8 secret cases one after another, each worth 10% of the points. Recall that your final score on the task is the score of your highest scoring submission.

Glowing Trunk

Time limit: 1 second | Memory limit: 1 gigabyte

As the Australian winter gets ever more frigid, the CSE students of UNSW have no choice but to leave their cold lonely homes and touch some grass. They discover a magnificent glowing tree trunk with no leaves that looks like this.

```
#
#
#
#
```

But wait… disaster strikes! The glowing tree is cut down but some evil engineering students. Can you help the poor CSE students rebuild the tree?

Input

You should read from standard input. We recommend using the templates at the top of the page to help you with input and output.

- The first and only line of input contains one integer N, the height of the glowing trunk.

Constraints

For all test cases:

- $1 \leq N \leq 100$.

Output

You should write to standard output.

- Output a single `#` on each line over N separate lines.

Sample Input 1

```
6
```

Sample Output 1

```
#
#
#
#
#
#
```

Explanation 1

In this case, we output a trunk of height 6.

Sample Input 2

```
1
```

Sample Output 2

```
#
```

Explanation 2

In this case, we output a tiny trunk of height 1.

Scoring

Your program will be run on the 2 sample cases and 8 secret cases one after another, each worth 10% of the points. Recall that your final score on the task is the score of your highest scoring submission.

Minion Missions

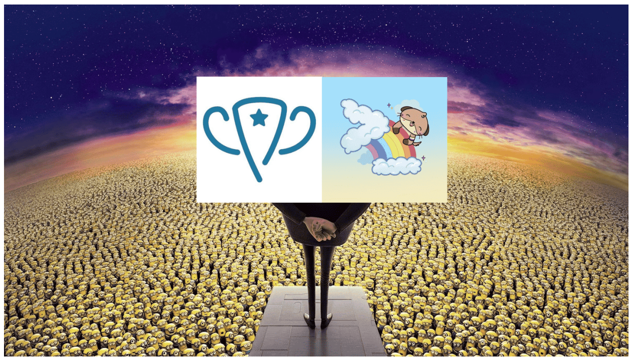
Time limit: 1 second | Memory limit: 1 gigabyte

CPMSoc and CSESoc have devised a plan to steal the moon.

To do this, they will send minions on a series of missions to conquer parts of the moon. In total, they have A_1 Andrés, B_1 Bobs, C_1 Carls and D_1 Daves available.

Each mission requires exactly A_2 Andrés, B_2 Bobs, C_2 Carls and D_2 Daves. No more, no less. You can't reuse minions for different missions, as they are very lazy.

What's the maximum number of missions that can occur?



Input

You should read from standard input. We recommend using the templates at the top of the page to help you with input and output.

- The first line of input contains the four integers A_1, B_1, C_1, and D_1.
- The second line of input contains the four integers A_2, B_2, C_2, and D_2.

Output

You should write to standard output.

- Output one integer, the maximum number of missions that can occur.

Constraints

For all test cases:

- $1 \le A_1, B_1, C_1, D_1 \le 100\,000$.
- $1 \le A_2, B_2, C_2, D_2 \le 100\,000$.

Sample Input 1

```
4 6 8 10
1 2 2 1
```

Sample Output 1

```
3
```

Explanation 1

In this case, there are 4 Andrés, 6 Bobs, 8 Carls and 10 Daves available.

Each mission requires 1 André, 2 Bobs, 2 Carls and 1 Dave.

Thus 3 missions can occur, using a total of 3 Andrés, 6 Bobs, 6 Carls and 3 Daves.

Sample Input 2

```
100 20 30 10
5 10 15 20
```

Sample Output 2

```
0
```

Explanation 2

In this case, they can't make any missions since there aren't enough Daves for even a single mission.

Scoring

Your program will be run on the 2 sample cases and 8 secret cases one after another, each worth 10% of the points. Recall that your final score on the task is the score of your highest-scoring submission.

Can you see the stage?

Time limit: 1 second | Memory limit: 1 gigabyte

You want to see your favorite artist, Sailor Twift. They are performing in a stadium with an N by M grid of seats.

You can't reserve a specific spot, so you want to make sure you can see the stage no matter where you sit. You can see the stage from a seat if its height is **strictly greater** than the height of every seat in front of it.

Can you see the stage from every seat?

Input

You should read from standard input. We recommend using the templates at the top of the page to help you with input and output.

- The first line of input contains two integers N and M, where N is the number of rows and M is the number of columns.
- Then follow N lines, each containing M integers. The ith line contains the heights of the seats in the ith row from the front, from left to right.

Output

You should write to standard output.

- Output `YES` if you can see the stage from every seat, and output `NO` otherwise.

Constraints

For all test cases:

- $1 \leq N, M \leq 1\,000$.
- The height of each seat is between 1 and 10^9 inclusive.

Sample Input 1

```
3 6
0 0 1 2 3 6
1 1 2 3 4 8
1 2 3 4 5 9
```

Sample Output 1

```
NO
```

Explanation 1

In the first column, there is a seat of height 1 behind a seat of height 1, so you cannot see the stage from this seat.

Sample Input 2

```
3 6
0 0 1 2 3 6
1 1 2 3 4 8
2 2 3 4 5 9
```

Sample Output 2

```
YES
```

Explanation 2

All seats have a higher height than every seat in front of them.

Scoring

Your program will be run on the 2 sample cases and 8 secret cases one after another, each worth 10% of the points. Recall that your final score on the task is the score of your highest-scoring submission.

Shiritori

Time limit: 1 second | Memory limit: 1 gigabyte

Kevin and Stuart are playing the famous Japanese word game, *Shiritori*. However one important prerequisite to playing this game is the ability to speak Japanese. Not to worry! They have devised a brand new edition, *Shiritori, Minionese Edition*. The rules are as follows.

- Player A begins the game by saying an Minionese word. This is easy, because any string of lowercase letters is a Minionese word.
- Player B responds with an Minionese word. **This word must be different from every word said before it, and must begin with the final letter of the previous word.**
- The game continues until a player makes a mistake. That player loses the game.

The rules seem quite straightforward however Kevin and Stuart are silly, and often don't notice when a player makes a mistake. Help our gamers find the first word which broke the rules. Or was the game played perfectly?

Input

You should read from standard input. We recommend using the templates at the top of the page to help you with input and output.

- The first line contains one integer N, the number of words in the game.
- Then follow N lines, each containing a word consisting of lowercase letters.

Output

You should write to standard output.

- If the game was played perfectly, output `-1`.
- Otherwise, output the first word that breaks the rules.

Constraints

For all test cases:

- $1 \leq N \leq 100$.
- Each word is made up of at least 1 and at most 20 lowercase letters.

Sample Input 1

```
6
banana
amsterdam
malingering
gratuitous
sabbatical
rastafarianism
```

Sample Output 1

```
rastafarianism
```

Explanation 1

The word "sabbatical" ends with an "l" but the next word "rastafarianism" does not begin with an "l". All other words follow the rules.

Sample Input 2

```
5
deuteronomy
ytterbium
murdered
deuteronomy
nascent
```

Sample Output 2

```
deuteronomy
```

Explanation 2

The fourth word "deuteronomy" is repeated, which is the first time a rule was broken. Note that the word "nascent" also breaks a rule.

Sample Input 3

```
6
umbrage
edification
neuroses
sophism
maniacal
locrian
```

Sample Output 3

```
-1
```

Explanation 3

Both players played perfectly. All words follow the rules of Shiritori.

Scoring

Your program will be run on the 3 sample cases and 7 secret cases one after another, each worth 10% of the points. Recall that your final score on the task is the score of your highest-scoring submission.

Peaked

Time limit: 1 second | Memory limit: 1 gigabyte

Given a sequence of N points representing the elevations of a mountain range from left to right, determine the number of peaks.

A peak is defined as a point, or a contiguous sequence of points of the same height, that is strictly higher than every point adjacent to it. See the sample cases for some examples.

Input

You should read from standard input. We recommend using the templates at the top of the page to help you with input and output.

- The first line of input contains one integer N, the number of points.
- The second line contains N integers, the elevations of the points from left to right.

Output

You should write to standard output.

- Output a single integer, the number of peaks in the sequence.

Constraints

For all test cases:

- $1 \leq N \leq 100\,000$.
- Each elevation is between 1 and 100 inclusive.

Additionally:

- For Subtask 1 (30% of points), $N \leq 100$ and all heights are distinct.
- For Subtask 2 (40% of points), $N \leq 100$.
- For Subtask 3 (30% of points), there are no additional constraints.

Sample Input 1

```
5
1 3 2 1 3
```

Sample Output 1

```
2
```

Explanation 1

There are two peaks: one at the second point with an elevation of 3, and another at the fifth point with an elevation of 3.

Sample Input 2

```
4
1 3 3 1
```

Sample Output 2

```
1
```

Explanation 2

There is only one peak: the flat top formed by the second and third points with an elevation of 3.

Scoring

For each subtask (worth 50%, 30% and 20% of points respectively, as per the Constraints section), your program will be run on multiple secret test cases one after another, and if it produces the correct output for **all** test cases, it solves that subtask. Your program will receive the points for each subtask it solves. Recall that your final score on the task is the score of your highest scoring submission.

Soup

Time limit: 1 second | Memory limit: 1 gigabyte

Terry the Turtle likes carrot soup.

Terry would like to make a soup with four different types. You have a certain number of carrots of each type, and each carrot has a certain tastiness value.

To make a balanced soup, Terry needs to use exactly K carrots in total, and for the ith type of carrot (where i is 1, 2, 3 or 4), Terry should use at least A_i and at most B_i carrots of that type.

Terry would like you to find the highest possible tastiness of any balanced soup he can make, where the tastiness of a soup is the sum of the tastiness of its ingredients. It is guaranteed that it is possible to make a balanced soup.

Input

You should read from standard input. We recommend using the templates at the top of the page to help you with input and output.

- The input contains nine lines. The first eight lines form four pairs.
  - The first line of the ith pair contains three integers N_i, A_i and B_i, where N_i is the number of that type of carrot, A_i is the minimum number of type i carrots in a balanced soup, and B_i is the maximum number of type i carrots in a balanced soup.
  - The second line of the ith pair contains N_i integers, denoting the tastiness values of the carrots of type i.
- The final line contains one integer K, the total number of carrots in a balanced soup.

Output

You should write to standard output.

- Output a single integer, the highest possible tastiness of any balanced soup Terry can make.

Constraints

For all test cases:

- $1 \leq A_i \leq B_i \leq N_i \leq 25\,000$ for all i.
- Each carrot tastiness is between 1 and 10 000 inclusive.
- $A_1 + A_2 + A_3 + A_4 \leq K \leq B_1 + B_2 + B_3 + B_4$.

Additionally:

- For Subtask 1 (40% of points): $N_i \leq 10$ for all i.
- For Subtask 2 (40% of points): $A_i = 1$ and $B_i = N_i$ for all i.
- For Subtask 3 (20% of points): there are no additional constraints.

Sample Input 1

```
2 1 2
80 40
10 1 3
23 39 3 60 7 60 90 1 70 90
5 2 5
7 10 4 7 9
9 7 9
4 7 1 3 6 8 2 1 9
13
```

Sample Output 1

```
388
```

Explanation 1

In this case, we can choose the following carrots:

- 1 carrot of type 1 with a tastiness value of 80.
- 3 carrots of type 2 with tastiness values of 90, 70 and 90.
- 2 carrots of type 3 with tastiness values of 10 and 9.
- 7 carrots of type 4 with tastiness values of 4, 7, 3, 6, 8, 2, and 9.

This satisfies the requires of a valid soup, since:

- There are $1 + 3 + 2 + 7 = 13 = K$ carrots in total.
- There are between 1 and 2 carrots of type 1.
- There are between 1 and 3 carrots of type 1.
- There are between 2 and 5 carrots of type 1.
- There are between 7 and 9 carrots of type 1.

The total tastiness is $80 + 90 + 70 + 90 + 10 + 9 + 4 + 7 + 3 + 6 + 8 + 2 + 9 = 388$.

Sample Input 2

```
10 3 6
1 2 3 4 5 6 7 8 9 10
10 3 6
1 2 3 4 5 6 7 8 9 10
10 4 7
1 2 3 4 5 6 7 8 9 10
5 2 5
20 22 24 26 28
16
```

Sample Output 2

```
215
```

Scoring

For each subtask (worth 40%, 40% and 20% of points respectively, as per the Constraints section), your program will be run on multiple secret test cases one after another, and if it produces the correct output for **all** test cases, it solves that subtask. Your program will receive the points for each subtask it solves. Recall that your final score on the task is the score of your highest scoring submission.

Diophantine Reciprocals

Time limit: 1 second | Memory limit: 1 gigabyte

You are given an integer n. Determine the number of distinct pairs of positive integers (x, y) that satisfy the equation:

$$\frac{1}{x} + \frac{1}{y} = \frac{1}{n}.$$

Two pairs (x_1, y_1) and (x_2, y_2) are considered distinct if $x_1 \neq x_2$ or $y_1 \neq y_2$. Note that $(1, 2)$ and $(2, 1)$ are considered distinct pairs.

Input

You should read from standard input. We recommend using the templates at the top of the page to help you with input and output.

The first and only line of input contains a single integer n.

Note that if you are using C, the value of n may exceed the maximum value of an `int`, so you should use the `long long` type. If you are using Python, you can ignore this.

Output

You should write to standard output.

- Output a single integer, the number of distinct pairs of positive integers (x, y) that satisfy the equation.

Constraints

- For Subtask 1 (50% of points), $1 \le n \le 10^3$.
- For Subtask 2 (30% of points), $1 \le n \le 10^6$.
- For Subtask 3 (20% of points), $1 \le n \le 10^{12}$.

Sample Input 1

```
4
```

Sample Output 1

```
5
```

Explanation 1

There are 5 distinct pairs of positive integers (x, y) that satisfy the equation: $(5, 20)$, $(6, 12)$, $(8, 8)$, $(12, 6)$, $(20, 5)$.

For example,

$$\frac{1}{5} + \frac{1}{20} = \frac{1}{4}.$$

Scoring

For each subtask (worth 50%, 30% and 20% of points respectively, as per the Constraints section), your program will be run on multiple secret test cases one after another, and if it produces the correct output for **all** test cases, it solves that subtask. Your program will receive the points for each subtask it solves. Recall that your final score on the task is the score of your highest scoring submission.

Grouping Students

Time limit: 1 second | Memory limit: 1 gigabyte

You are a teacher organising a group activity for your students. In your class, each student is either of type 1, type 2 or type 3. A student of type x will be happy if the size of their group is a multiple of x.

Given the amount of students of each type, is it possible to divide them into groups so that they are all happy? If so, output a possible list of groups.

Input

You should read from standard input. We recommend using the templates at the top of the page to help you with input and output.

- The first and only line of input contains three integers A, B and C, the number of students who want the size of their group to be a multiple of 1, 2 and 3 respectively.

Output

You should write to standard output.

If it is not possible for all students to be happy, output `-1`.

Otherwise, output a list of groups. In particular:

- The first line of output contains one integer G, the number of groups.
- Then follow G lines, the ith of which contains an integer k_i, the size of the ith group, followed by k_i integers, the types of the students in the ith group (in any order).

If there are multiple correct outputs, any one of them will be accepted.

Constraints

For all test cases:

- $0 \leq A, B, C \leq 100\,000$.

Additionally:

- For Subtask 1 (20% of points): $C = 0$.
- For Subtask 2 (30% of points): $A = 0$.
- For Subtask 3 (30% of points): $A, B, C \leq 100$.
- For Subtask 4 (20% of points): there are no additional constraints.

Sample Input 1

```
1 3 0
```

Sample Output 1

```
2
2 1 2
2 2 2
```

This shows one possible list of groups. The first group contains a student of type 1 and a student of type 2. The second group contains two students of type 2. All the type 2 students are happy because the size of their group is even. Type 1 students are always happy.

Sample Input 2

```
1 1 4
```

Sample Output 2

```
1
6 3 1 2 3 3 3
```

Explanation 2

In this case, we can create a single group of size 6, which is a multiple of 1, 2 and 3.

Sample Input 3

```
0 1 0
```

Sample Output 3

```
-1
```

Explanation 3

In this case, there is one student who wants to be in an even-sized group, which is impossible.

Scoring

For each subtask (worth 20%, 30%, 30% and 20% of points, as per the Constraints section), your program will be run on multiple secret test cases one after another, and if it produces the correct output for **all** test cases, it solves that subtask. Your program will receive the points for each subtask it solves. Recall that your final score on the task is the score of your highest scoring submission.