Hong Kong Olympiad in Informatics 2025/26 Junior Group

Task Overview

ID	Name	Time Limit	Memory Limit	Subtasks
J261	Cubic Date	1.000 s	256 MB	12 + 19 + 14 + 27 + 21 + 7
J262	Bowling Score Recovery	1.000 s	256 MB	10 + 13 + 14 + 17 + 18 + 28
J263	Spot-Check Debugging	1.500 s	256 MB	10 + 8 + 13 + 19 + 22 + 28
J264	Shy Students	1.000 s	256 MB	9 + 8 + 17 + 19 + 20 + 27

Notice:
Unless otherwise specified, inputs and outputs shall follow the format below:

- One space between a number and another number or character in the same line.
- No space between characters in the same line.
- Each string shall be placed in its own separate line.
- Outputs will be automatically fixed as follows: Trailing spaces in each line will be removed and an end-of-line character will be added to the end of the output if not present. All other format errors will not be fixed.

C++ programmers should be aware that using C++ streams (`cin` / `cout`) may lead to I/O bottlenecks and substantially lower performance.

For some problems 64-bit integers may be required. In C++ it is `long long` and its token for `scanf`/`printf` is `%lld`.

All tasks are divided into subtasks. You need to pass all test cases in a subtask to get points.

J261 - CUBIC DATE

Time Limit: 1.000 s / Memory Limit: 256 MB

Alice loves cubes. This year, she received 4 cubes as her birthday present! Each of the cubes has the same digit written on all 6 faces of the cube.

Alice uses D_1, D_2, D_3 and D_4 to denote the digits written on the 4 cubes. For $1 \leq i \leq 4$, the value of D_i satisfies $0 \leq D_i \leq 9$.

As the year 2026 is approaching, Alice wishes to use all 4 cubes to display a date in 2026. She must use exactly 2 cubes to display the month, and the other 2 cubes to display the day. The displayed date must be valid, that is, the month of the date must be between 1 and 12 (inclusive), the day number must be at least 1 and not exceeding the number of days of that month. The number of days of each month in 2026 is shown in the following table.

Month	1	2	3	4	5	6	7	8	9	10	11	12
Number of Days	31	28	31	30	31	30	31	31	30	31	30	31

For example, if the digits written on the 4 cubes are 1, 1, 2 and 3 respectively, Alice can display the date `12-31`, i.e. the 31-st of December. However, if the digits written are 1, 9, 9 and 9 respectively, there is no valid date that Alice can display.

Given the digits D_1, D_2, D_3 and D_4, please help Alice to display a valid date in 2026, or determine if it is impossible to do so. Note that the cubes cannot be rotated or flipped, for example, it is prohibited to use a cube with digit 6 written on to display the digit 9, or vice versa.

INPUT

The input consists of 4 lines. The i-th line contains a single digit D_i, the digit written on the i-th cube.

It is guaranteed that the digits written on the cubes are given in **non-decreasing order**, that is, $D_1 \leq D_2 \leq D_3 \leq D_4$.

OUTPUT

If it is impossible to display a valid date with the digits D_1, D_2, D_3 and D_4, output `No`.

Otherwise, output `Yes` on the first line. Output a date in the format `MM-DD` on the second line, where `MM` and `DD` are two-digit numbers (possibly containing leading zeros) representing the month and the day of the date respectively.

If there are multiple possible valid dates that can be displayed, you may output any of them.

Output Format

Outputs are **case sensitive**, for example, the outputs `no` and `NO` will be regarded as incorrect.

SAMPLE TESTS

	Input	Output
1	1 1 2 3	Yes 12-31

Other acceptable answers include `11-23` and `12-13`. However, both `11-32` and `31-12` are not acceptable.

	Input	Output
2	0 2 2 9	Yes 09-22

Note that the date `02-29` does **not** exist in year 2026.

3	0 0 0 0	No
4	1 9 9 9	No
5	0 0 1 1	Yes 01-01

SUBTASKS

For all cases:
$0 \leq D_1 \leq D_2 \leq D_3 \leq D_4 \leq 9$

	Points	Constraints
1	12	$D_1 = D_2 = 0$ $D_3 \neq 0$ $D_4 \neq 0$
2	19	$D_1 = D_2 = 0$
3	14	$D_1 = 0$ $D_2 = 1$ $D_3 \geq 1$ $D_4 \geq 1$
4	27	$D_1 = 0$
5	21	$D_1 = 1$ $D_2 \geq 1$ $D_3 \geq 1$ $D_4 \geq 1$
6	7	No additional constraints

J262 - BOWLING SCORE RECOVERY

Time Limit: 1.000 s / Memory Limit: 256 MB

Alice is training to become a professional bowling athlete! She intends to join the Byteland Bowling League, which uses the following ruleset:

The game will consist of an arbitrary amount of *frames*. Each frame has 10 bowling pins that Alice needs to knock down using up to two bowling balls. The scores that the player will get for the frame will vary according to the outcome, as listed below:

- Strike: all 10 pins are knocked down by the first bowling ball of the frame. The player is rewarded 30 points, and the next frame will then begin.
- Spare: all 10 pins are knocked down by two bowling balls and the first ball did not knock down all pins. The player is rewarded $10 + x$ points, where x is the number of pins knocked down by the first bowling ball of the frame.
- Open frame: there are still pins remaining after both bowling balls are bowled. The player is rewarded with y points, where y is the total number of pins knocked down by the two bowling balls of the frame.

For example, if Alice knocks down all 10 pins with the first ball, it counts as a Strike and Alice will get 30 points. If Alice knocks down 3 pins with the first ball and 7 with the second ball, it counts as a Spare and Alice will get 10 + 3 = 13 points.

However, the scoring system of the bowling alley has malfunctioned! The system only recorded the pin count of the N bowling balls that **knocked down at least one pin**. Since Alice already did a lot of practice and is tired, she asks you to calculate the maximum possible score that she may have gotten.

INPUT

The first line of the input consists of a single integer N, the number of bowling ball throws recorded.

The next N lines consist of an integer A_i, the number of pins knocked down by the i-th recorded bowling ball.

OUTPUT

The output consists of a single integer, the maximum possible score that Alice may have gotten.

SAMPLE TESTS

	Input	Output
1	4 10 3 6 4	49

One of the possible sequences of pins knocked down is (10, 3, 0, 6, 4). Note that the ball with 0 pins was not recorded by the system.
In this scenario, the score Alice would have gotten will be as follows:

- The first ball will count as a Strike and Alice gets 30 points.
- The next two balls will count as Open Frame and Alice gets 3 + 0 = 3 points.
- The last two balls will count as a Spare and Alice gets 10 + 6 = 16 points.

Therefore Alice will get 30 + 3 + 16 = 49 points in total.
It can be proven that this is the maximum score that Alice may have scored.

	Input	Output
2	4 7 3 6 7	30

One of the possible sequences of pins knocked down to achieve the maximum is (7, 3, 6, 0, 7, 0).
Note that (7, 3, 6, 7) is not a valid sequence, as the second frame will have 6 + 7 = 13 pins knocked down, exceeding the maximum possible amount of 10.

3	5 8 2 8 2 8	44

One of the possible sequences of pins knocked down to achieve the maximum is (8, 2, 8, 2, 8, 0).

SUBTASKS

For all cases:
$1 \leq N \leq 10^5$
$1 \leq A_i \leq 10$ for $1 \leq i \leq N$

	Points	Constraints
1	10	$N = 2$ $A_1 \neq 10$ $A_2 \neq 10$
2	13	$A_i < 5$ for $1 \leq i < N$
3	14	$A_i + A_{i+1} \neq 10$ for $1 \leq i < N$
4	17	$N \geq 2$ $A_i + A_{i+1} = 10$ for $1 \leq i < N$ $A_1 > A_2$
5	18	$A_i + A_{i+1} = 10$ for $1 \leq i < N$
6	28	No additional constraints

J263 - SPOT-CHECK DEBUGGING

Time Limit: 1.500 s / Memory Limit: 256 MB

Bob works at Heung Shing Technology. The company has a legacy code base with L lines of code, indexed from 1 to L.

The code contains N disjoint buggy sections. The i-th section covers a continuous range of lines from U_i to V_i (inclusive). Initially, all lines in these ranges are buggy, and all other lines are bug-free. The sections satisfy $1 \le U_i \le V_i \le L$ for all $1 \le i \le N$. Moreover, they are sorted, and each two buggy sections are separated by at least one bug-free line, meaning $V_i + 1 < U_{i+1}$ for all $1 \le i < N$.

Bob performs a routine every day, starting from day 1. On each day, Bob has a starting offset t (a line number). He inspects lines $t, t+S, t+2S, \ldots$ in this order, where S is a fixed step size. He stops inspecting immediately when he finds a line X that is currently buggy. Bob is smart so he can always tell whether a line is buggy. If he inspects past line L without finding any bugs, he fixes no code on that day.

After finding the first buggy line X, Bob enters debug mode. He fixes line X (so the line becomes bug-free), then fixes line $X-1$, then line $X-2$, and so on. He continues fixing lines backwards until he reaches a bug-free line or until he fixes line 1, then he stops working for that day. Note that once a line is fixed, it remains bug-free in all subsequent days.

For example, suppose the step size is $S = 3$, the starting offset is $t = 3$, and initially lines 4, 6-11, and 14-15 are buggy. Bob inspects lines $3, 6, 9, 12, 15, \ldots$ in order. Line 6 is the first buggy line he encounters, so $X = 6$.

Lines 1-3			Lines 4-6			Lines 7-9			Lines 10-12			Lines 13-15		
0	0	0	1	0	1	1	1	1	1	1	0	0	1	1

Underscored/blue cells show lines Bob inspected.

Bob then fixes line 6, but stops immediately because line 5 is already bug-free. Thus, Bob fixes 1 line on this day.

Lines 1-3			Lines 4-6			Lines 7-9			Lines 10-12			Lines 13-15		
0	0	0	1	0	**0**	1	1	1	1	1	0	0	1	1

Bold/green cells show lines Bob fixed.

Bob uses a cyclic schedule to determine the starting offset for each day. He has a sequence of P starting offsets $T_1, T_2, \ldots, T_P$ (where $1 \le T_i \le S$). On the first P days, Bob uses offsets $T_1, T_2, \ldots, T_P$ respectively. On day $P+1$, the schedule cycles back, so Bob uses T_1 again, and so on. Formally, on day D, the offset is T_k, where $k = ((D-1) \bmod P) + 1$.

Continuing the example above, suppose Bob has the offset sequence $T_1 = 3, T_2 = 1$ (so $P = 2$). It means that Bob uses the offset $T_1 = 3$ on days 1, 3, 5 etc., and $T_2 = 1$ on day 2, 4, 6 etc. After day 1 (where Bob fixed line 6 as described), lines 4, 6-11, and 14-15 remain buggy. On day 2, Bob uses offset $T_2 = 1$. He inspects lines $1, 4, 7, 10, 13, \ldots$ in order. Line 4 is the first buggy line. Bob fixes line 4, but stops immediately because line 3 is bug-free. Thus, Bob fixes 1 line. On day 3, the schedule cycles back, so Bob uses offset $T_1 = 3$ again. He inspects lines $3, 6, 9, 12, 15, \ldots$ Line 9 is the first buggy line. Bob then fixes lines 9, 8, 7, fixing 3 lines total, and so on.

The tables below illustrate the complete fixing process for this example from day 2 onwards.

Day 2: Starting offset $T_2 = 1$, 1 line fixed

Step	Lines 1-3			Lines 4-6			Lines 7-9			Lines 10-12			Lines 13-15		
After inspection	0	0	0	1	0	0	1	1	1	1	1	0	0	1	1
After fixing	0	0	0	**0**	0	0	1	1	1	1	1	0	0	1	1

Day 3: Starting offset $T_1 = 3$, 3 lines fixed

Step	Lines 1-3			Lines 4-6			Lines 7-9			Lines 10-12			Lines 13-15		
After inspection	0	0	0	0	0	0	1	1	1	1	1	0	0	1	1
After fixing	0	0	0	0	0	0	**0**	**0**	**0**	1	1	0	0	1	1

Day 4: Starting offset $T_2 = 1$, 0 lines fixed

Step	Lines 1-3			Lines 4-6			Lines 7-9			Lines 10-12			Lines 13-15		
After inspection	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1
After fixing	0	0	0	0	0	0	0	0	0	**0**	1	0	0	1	1

Day 5: Starting offset $T_1 = 3$, 2 lines fixed

Step	Lines 1-3			Lines 4-6			Lines 7-9			Lines 10-12			Lines 13-15		
After inspection	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1
After fixing	0	0	0	0	0	0	0	0	0	0	1	0	0	**0**	**0**

Day 6: Starting offset $T_2 = 1$, 0 lines fixed

Step	Lines 1-3			Lines 4-6			Lines 7-9			Lines 10-12			Lines 13-15		
After inspection	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
After fixing	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

No more fixing is done from day 6 onwards. Line 11 remains buggy indefinitely because Bob's inspection pattern (with offsets 3 and 1, step size 3) never checks line 11.

You need to answer Q queries. The i-th ($1 \le i \le Q$) query gives a day D_i. You must calculate the **number of lines fixed** on that specific day. Note that the bug states persist and evolve from day 1 onwards, even if a specific day is not queried.

INPUT

The first line contains two integers N and L.

The next N lines describe the initial buggy sections. The i-th line ($1 \le i \le N$) contains two integers U_i and V_i.

The next line contains two integers S and P.

The next line contains P integers $T_1, T_2, \ldots, T_P$.

The next line contains a single integer Q.

The next line contains Q integers $D_1, D_2, \ldots, D_Q$.

OUTPUT

Output a single line containing Q integers separated by spaces. The i-th integer is the number of lines Bob fixes on day D_i.

SAMPLE TESTS

1

Input	Output
3 15 4 4 6 11 14 15 3 2 3 1 5 1 3 5 6 100	1 3 2 0 0

Refer to the problem statement.

2

Input	Output
3 25 5 9 12 15 20 23 6 3 2 3 2 6 1 2 3 5 8 10	4 1 3 1 1 0

On day 1, Bob fixes lines 5 to 8.
On day 2, Bob fixes line 9.
On day 3, Bob fixes lines 12 to 14.
On day 4, Bob fixes line 20.
On day 5, Bob fixes line 15.
On day 6 and 7, Bob fixes no lines.
On day 8, Bob fixes line 21.
From day 9 onwards, there are no line fixes.

3

Input	Output
4 67 7 26 37 37 40 42 56 66 10 4 6 7 10 7 11 1 2 3 4 5 6 7 8 9 10 11	10 1 3 1 6 2 1 0 9 0 0

4

Input	Output
1 24 1 19 5 3 2 1 5 10 1 2 3 4 5 6 7 8 9 10	2 4 4 2 4 0 1 0 0 0

SUBTASKS

For all cases:
$1 \le N \le 1500$
$1 \le L \le 10^9$
$1 \le U_i \le V_i \le L$ for $1 \le i \le N$
$V_i + 1 < U_{i+1}$ for $1 \le i \le N - 1$
$1 \le S \le L$
$1 \le P \le 1500$
$1 \le T_i \le S$ for $1 \le i \le P$
$1 \le Q \le 10^5$
$1 \le D_i < D_{i+1} \le 10^9$ for $1 \le i \le Q - 1$

	Points	Constraints
1	10	$L \le 100$ $D_Q \le 100$
2	8	$P = 1$ $U_i = V_i$ for $1 \le i \le N$
3	13	$P = 1$ $V_i - U_i + 1 \le S$ for all $1 \le i \le N$
4	19	$P = 1$
5	22	$N = 1$
6	28	No additional constraints

J264 - SHY STUDENTS

Time Limit: 1.000 s / Memory Limit: 256 MB

As a new teacher of Byteland Secondary School, you realise that your designated class is comprised entirely of shy students. As a result, it proves to be a nightmare to design a seating plan to satisfy everyone...

In Byteland Secondary School, all classrooms are grids with 4000 rows and 4000 columns. As a teacher, you know that you realistically may not need all this space to seat your students. Therefore, your classroom seating plan does not have to use all 4000 rows and columns, and some of the cells may be kept empty.

Figure 1: Two cells are adjacent if they share an edge

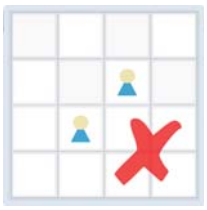

Figure 2: Two cells are **not** adjacent if they only share a corner

In your seating plan, two cells (r_1, c_1) and (r_2, c_2) are *adjacent* if they share an edge. Formally, this is expressed as $|r_1 - r_2| + |c_1 - c_2| = 1$.

Your class has N students in total. In a seating plan, each of the N students is designated a unique cell on the classroom grid. In order for the seating plan to be considered **valid**, the students have to be *connected*. The students are *connected* if and only if any two students can reach each other's cells only by repeatedly moving to adjacent cells occupied by students.

Figure 3: The students are connected

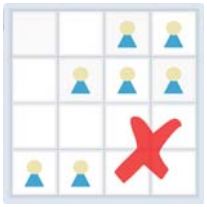

Figure 4: The students are **not** connected

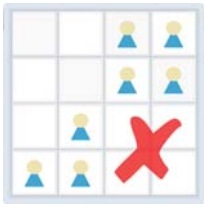

Figure 5: The students are **not** connected

Each student has a specific level of shyness s where $1 \leq s \leq 4$. A student with a shyness level s is only satisfied if they sit adjacent to at most s other students. For example, a student with shyness level 3 may sit next to 0, 1, 2, or 3 other students, but not 4 other students.

Currently, you know that there are N_1, N_2, N_3, and N_4 (where $N_1 + N_2 + N_3 + N_4 = N$) students of shyness value 1, 2, 3, and 4 respectively. You are tasked to decide on the dimensions $R \times C$ and design a valid seating plan that contain all N students' cells, such that all the shy students are satisfied. Note that R and C must satisfy $1 \leq R, C \leq 4000$. It may be possible that no such seating plan exists, in which you should report it.

Figure 6: A student with shyness level 3 can sit adjacent to at most 3 other students

Figure 7: A student with shyness level 3 can sit adjacent to at most 3 other students

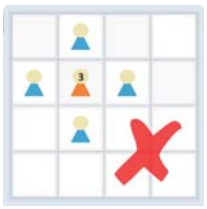

Figure 8: A student with shyness level 3 **cannot** sit adjacent to more than 3 other students

INPUT

The first and only line of input consists of four integers N_1, N_2, N_3 and N_4.

OUTPUT

If there does not exist a valid seating plan to satisfy all the students, output `Impossible`.

Otherwise, output `Possible` on the first line. Output the dimensions of the seating plan R and C on the second line. Then output R lines, each containing C characters, representing the seating plan.

In a seating plan, the character `.` represents a cell not occupied by any student, and the characters `1`, `2`, `3`, `4` represent the shyness value of the student seated at that cell.

If there are multiple seating plans, you may output any of them. Note that the dimensions of your seating plan do not have to be minimised, and only have to satisfy $1 \leq R, C \leq 4000$.

SAMPLE TESTS

	Input	Output

1	`3 3 1 2`	`Possible`
		`2 8`
		`....1...`
		`12223441`

The student with shyness value 3 is adjacent to 3 other students, so the student is satisfied.
Both students with shyness value 4 are adjacent to only 2 other students, which does not exceed 4, so they are also satisfied.
It can be seen that all the students are satisfied, hence the seating plan is valid.

2	`2 1 7 7`	`Possible`
		`4 7`
		`.444...`
		`244441.`
		`.33.3..`
		`13333..`

3	`4 1 0 0`	`Impossible`

It can be proven that there does not exist a seating plan such that all students are simultaneously satisfied and connected.

SUBTASKS

For all cases:
$0 \leq N_1, N_2, N_3, N_4 \leq 1000$
$1 \leq N_1 + N_2 + N_3 + N_4 \leq 4000$

	Points	Constraints
1	9	$N_1 = 0$
2	8	$N_1 \leq 2$
3	17	$N_2 = N_3 = 0$
4	19	$N_3 = 0$
5	20	$N_2 = 0$
6	27	No additional constraints