Problem J1: Conveyor Belt Sushi

Problem Description

There is a new conveyor belt sushi restaurant in town. Plates of sushi travel around the restaurant on a raised conveyor belt and customers choose what to eat by removing plates.

Each red plate of sushi costs $3, each green plate of sushi costs $4, and each blue plate of sushi costs $5.

Your job is to determine the cost of a meal, given the number of plates of each colour chosen by a customer.

Input Specification

The first line of input contains a non-negative integer, R, representing the number of red plates chosen. The second line contains a non-negative integer, G, representing the number of green plates chosen. The third line contains a non-negative integer, B, representing the number of blue plates chosen.

Output Specification

Output the non-negative integer, C, which is the cost of the meal in dollars.

Sample Input

```
0
2
4
```

Output for Sample Input

```
28
```

Explanation of Output for Sample Input

This customer chose 0 red plates, 2 green plates, and 4 blue plates. Therefore, the cost of the meal in dollars is $0 \times 3 + 2 \times 4 + 4 \times 5 = 28$.

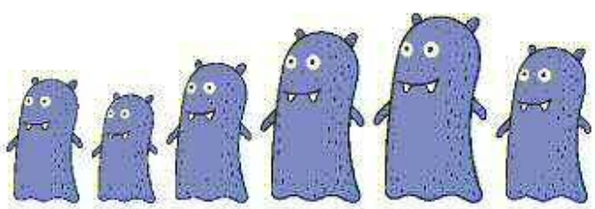
Problem J2: Dusa And The Yobis

Problem Description
Dusa eats Yobis, but only Yobis of a certain size.

If Dusa encounters a Yobi that is smaller than itself, it eats the Yobi, and absorbs its size. For example, if Dusa is of size 10 and it encounters a Yobi of size 6, Dusa eats the Yobi and expands to size $10 + 6 = 16$.

If Dusa encounters a Yobi that is the same size as itself or larger, Dusa runs away without eating the Yobi.

Dusa is currently facing a line of Yobis and will encounter them in order. Dusa is guaranteed to eventually encounter a Yobi that causes it to run away. Your job is to determine Dusa's size when this happens.



Input Specification
The first line of input contains a positive integer, D, representing Dusa's starting size.

The remaining lines of input contain positive integers representing the sizes of the Yobis in order.

Output Specification
Output the positive integer, R, which is Dusa's size when it eventually runs away.

Sample Input 1
```
5
3
2
9
20
22
14
```

Output for Sample Input 1
```
19
```

Explanation of Output for Sample Input 1
Dusa is large enough to eat the Yobi of size 3. This brings Dusa's size to 8. Dusa is large enough to eat the Yobi of size 2. This brings Dusa's size to 10. Dusa is large enough to eat the Yobi of size 9. This brings Dusa's size to 19. The Yobi of size 20 causes Dusa to run away.

Sample Input 2
```
10
10
3
5
13
```

Output for Sample Input 2
```
10
```

Explanation of Output for Sample Input 2
The Yobi of size 10 causes Dusa to run away, leaving its size unchanged.

Problem J3: Bronze Count

Problem Description
After completing a competition, you are struck with curiosity. How many participants were awarded bronze level?

Gold level is awarded to all participants who achieve the highest score. Silver level is awarded to all participants who achieve the second highest score. Bronze level is awarded to all participants who achieve the third highest score.

Given a list of all the scores, your job is to determine the score required for bronze level and how many participants achieved this score.

Input Specification
The first line of input contains a positive integer, N, representing the number of participants.

Each of the next N lines of input contain a single integer representing a participant's score.

Each score is between 0 and 75 (inclusive) and there will be at least three distinct scores.

The following table shows how the available 15 marks are distributed:

Marks	Description	Bound
6	The scores are distinct and the number of participants is small.	$N \leq 50$
7	The scores might not be distinct and the number of participants is small.	$N \leq 50$
2	The scores might not be distinct and the number of participants could be large.	$N \leq 250\,000$

Output Specification
Output a non-negative integer, S, and a positive integer, P, separated by a single space, where S is the score required for bronze level and P is how many participants achieved this score.

Sample Input 1
```
4
70
62
58
73
```

Output for Sample Input 1
```
62 1
```

Explanation of Output for Sample Input 1
The score required for bronze level is 62 and one participant achieved this score.

Sample Input 2
```
8
75
70
60
70
70
60
75
70
```

Output for Sample Input 2
```
60 2
```

Explanation of Output for Sample Input 2
The score required for bronze level is 60 and two participants achieved this score.

Problem J4: Troublesome Keys

Problem Description

As Alex is typing, their keyboard is acting strangely. Two letter keys are causing trouble:

- One letter key displays the same wrong letter each time it is pressed. Alex calls this key the *silly key*. Oddly, Alex never actually tries to type the wrong letter displayed by the silly key.
- Another letter key doesn't display anything when it is pressed. Alex calls this key the *quiet key*.

Alex presses the silly key at least once but they don't necessarily press the quiet key.

Your job is to determine the troublesome keys and the wrong letter that is displayed. Luckily, this is possible because Alex never presses the silly key immediately after pressing the quiet key and Alex never presses the quiet key immediately after pressing the silly key.

Input Specification

There will be two lines of input. The first line of input represents the N keys Alex presses on the keyboard. The second line of input represents the letters displayed on the screen.

Both lines of input will only contain lowercase letters of the alphabet.

The following table shows how the available 15 marks are distributed.

Marks	Description	Bound
3	The quiet key is not pressed. A small number of keys are pressed.	$N \leq 50$
3	The first troublesome key pressed is the silly key. A small number of keys are pressed.	$N \leq 50$
5	The first troublesome key pressed may be the silly key or the quiet key. A small number of keys are pressed.	$N \leq 50$
4	The first troublesome key pressed may be the silly key or the quiet key. A large number of keys are pressed.	$N \leq 500\,000$

Output Specification

There will be two lines of output.

On the first line, output the letter corresponding to the silly key and the wrong letter displayed on the screen when it is pressed, separated by a single space.

On the second line, output the letter corresponding to the quiet key if it is pressed. Output the dash character (-) if the quiet key is not pressed.

Sample Input 1

```
forloops
fxrlxxps
```

Output for Sample Input 1

```
o x
-
```

Explanation of Sample Output 1

The letter corresponding to the silly key was the letter `o`. Each time it was pressed, the wrong letter `x` was displayed. The quiet key was not pressed.

Sample Input 2

```
forloops
fxrlxxp
```

Output for Sample Input 2

```
o x
s
```

Explanation of Sample Output 2

The letter corresponding to the silly key was the letter `o`. Each time it was pressed, the wrong letter `x` was displayed. The quiet key corresponds to the letter `s` which was not displayed.

Sample Input 3

```
forloops
frlpz
```

Output for Sample Input 3

```
s z
o
```

Explanation of Sample Output 3

The letter corresponding to the silly key was the letter `s`. Each time it was pressed, the wrong letter `z` was displayed. The quiet key corresponds to the letter `o` which was not displayed.

Problem J5: Harvest Waterloo

Problem Description

There is a wildly popular new harvest simulation game called *Harvest Waterloo*. The game is played on a rectangular pumpkin patch which contains bales of hay and pumpkins of different sizes. To begin the game, a farmer is placed at the location of a pumpkin.

The farmer harvests all pumpkins they can reach by moving left, right, up, and down throughout the patch. The farmer cannot move diagonally. The farmer can also not move through a bale of hay nor move outside of the patch.

Your job is to determine the total value of all the pumpkins harvested by the farmer. A small pumpkin is worth \$1, a medium pumpkin is worth \$5, and a large pumpkin is worth \$10 dollars.

Input Specification

The first line of input is an integer $R > 0$ which is the number of rows within the patch.

The second line of input is an integer $C > 0$ which is the number of columns within the patch.

The next R lines describe the patch. Each line will contain C characters and each character will either represent a pumpkin size or a bale of hay: `S` for a small pumpkin, `M` for a medium pumpkin, `L` for a large pumpkin, or $*$ for a bale of hay.

The next line of input is an integer A where $0 \leq A < R$, and the last line of input is an integer B where $0 \leq B < C$. Row A and column B is the starting location of the farmer and the top-left corner of the patch is row 0 and column 0.

The following table shows how the available 15 marks are distributed:

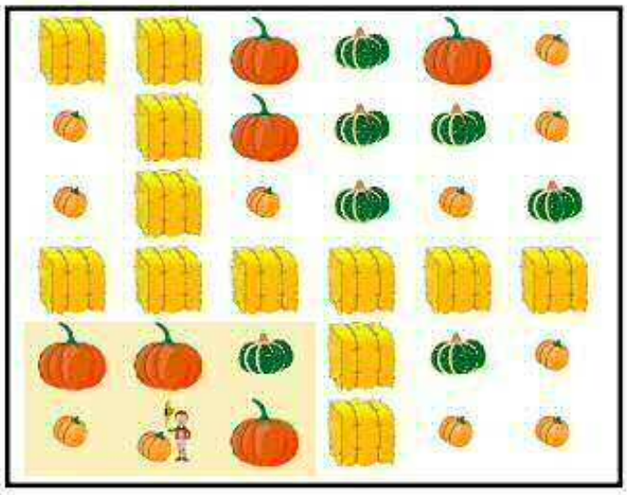
Marks	Description	Bound
1	The patch is small and there are no bales of hay.	$R \times C \leq 100$
4	The patch is small and the bales of hay divide the entire patch into rectangular areas.	$R \times C \leq 100$
5	The patch is small and the bales of hay can be anywhere.	$R \times C \leq 100$
5	The patch is large and the bales of hay can be anywhere.	$R \times C \leq 100\,000$

Output Specification

Output the integer, V, which is the total value in dollars of all the pumpkins harvested by the farmer.

Sample Input 1

```
6
6
**LMLS
S*LMMS
S*SMSM
******
LLM*MS
SSL*SS
5
1
```



Output for Sample Input 1

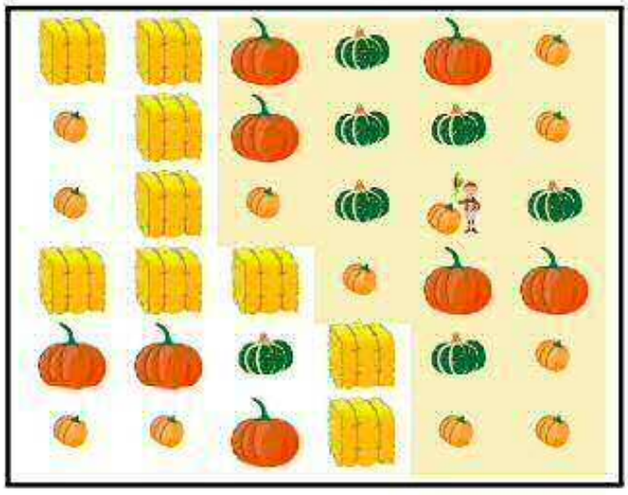
```
37
```

Explanation of Output for Sample Input 1

Starting at row 5 and column 1, the farmer can reach the 6 pumpkins in the highlighted area. They harvest 2 small pumpkins, 1 medium pumpkin, and 3 large pumpkins. The total value in dollars of this harvest is $2 \times 1 + 1 \times 5 + 3 \times 10 = 37$.

Sample Input 2

```
6
6
**LMLS
S*LMMS
S*SMSM
***SLL
LLM*MS
SSL*SS
2
4
```



Output for Sample Input 2

```
88
```

Explanation of Output for Sample Input 2

Starting at row 2 and column 4, the farmer can reach the 19 pumpkins in the highlighted area. They harvest 8 small pumpkins, 6 medium pumpkin, and 5 large pumpkins. The total value in dollars of this harvest is $8 \times 1 + 6 \times 5 + 5 \times 10 = 88$.

Problem S1: Hat Circle

Problem Description
At a recent social gathering, N people sit around a circular table, where N is even. The seats are numbered clockwise from 1 to N. Each person is wearing a hat with a number on it. Specifically, the person at seat i is wearing a hat with the number H_i on it.

Each person looks at the person who is directly across (diametrically opposite) them in the circle.

Determine the number of people who see someone with a hat with the same number as their own.

Input Specification
The first line of input will consist of one even positive integer N, representing the number of people at the social gathering.

The next N lines each contain a single non-negative integer H_i, representing the hat number of person i.

The following table shows how the available 15 marks are distributed:

Marks	**Description**	**Bounds on N**	**Bounds on H_i**
2	Very small number of people; only two hat numbers	$N \leq 4$	$H_i \leq 1$
1	Only one hat number	$N \leq 100$	$H_i = 1$
2	People in even numbered seats have hat number 1; people in odd numbered seats have hat number 0	$N \leq 100$	$H_i \leq 1$
5	Medium number of people	$N \leq 2\,000$	$H_i \leq 4\,000$
5	Large number of people and hat numbers	$N \leq 1\,000\,000$	$H_i \leq 2\,000\,000$

Output Specification
Output a single integer representing the number of people who see their hat number on the person directly across from them.

Sample Input 1
```
4
0
1
0
1
```

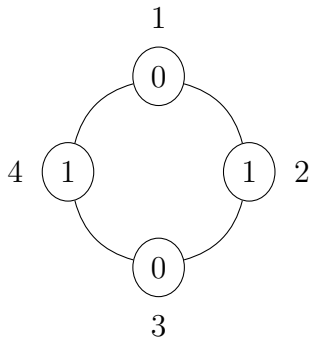
La version française figure à la suite de la version anglaise.

Output for Sample Input 1

```
4
```

Explanation of Output for Sample Input 1

The four seats around the table are shown below. Hat numbers are shown inside each seat and seat numbers are shown beside each seat. Notice that every person sees their hat number. The people in seats 1 and 3 both see hat number 0, and the people in seats 2 and 4 both see hat number 1.



Sample Input 2

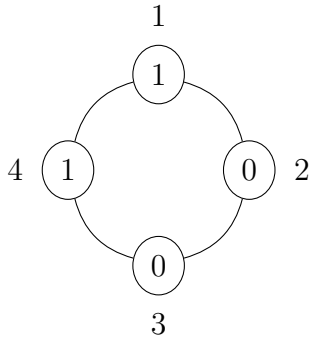
```
4
1
0
0
1
```

Output for Sample Input 2

```
0
```

Explanation of Output for Sample Input 2

The four seats around the table are shown below. Hat numbers are shown inside each seat and seat numbers are shown beside each seat. Notice that no person sees their hat number. The people in seats 1 and 4 both see hat number 0, and the people in seats 2 and 3 both see hat number 1.



La version française figure à la suite de la version anglaise.

Problem S2: Heavy-Light Composition

Problem Description
In a string containing only lowercase letters of the alphabet ("a" through "z"), we say a letter is *heavy* if it appears more than once in the string, and *light* otherwise.

We will be given a number of strings. For each string, we would like to determine whether the letters of the string alternate between light and heavy.

Input Specification
The first line of input will consist of two positive integers T and N, representing the number of strings and the length of each string.

The next T lines each contain a sequence of N lowercase letters of the alphabet.

The following table shows how the available 15 marks are distributed:

Marks	**Bounds on T**	**Bounds on N**	**Other Restrictions**
5	$2 \leq T \leq 4$	$2 \leq N \leq 4$	Only the letters "a" and "b" will be used
5	$2 \leq T \leq 10$	$2 \leq N \leq 30$	None
2	$2 \leq T \leq 100$	$2 \leq N \leq 100$	Only the letter "a" will be heavy; all other letters are light
3	$2 \leq T \leq 10\,000$	$2 \leq N \leq 100$	None

Output Specification
Output T lines, where each line will be either `T` or `F`. If the i-th input string does alternate between light and heavy letters, the i-th line of output should be `T`; and otherwise, the i-th line of output should be `F`.

Sample Input 1
```
3 4
abcb
bcbb
babc
```

Output for Sample Input 1
```
T
F
T
```

La version française figure à la suite de la version anglaise.

Explanation of Output for Sample Input 1
The first string is composed of a light letter, then a heavy letter, then a light letter, and then a heavy letter.

The second string ends in two consecutive heavy letters.

The third string is composed of a heavy letter, then a light letter, then a heavy letter, and then a light letter.

Sample Input 2

```
2 3
abc
bcb
```

Output for Sample Input 2

```
F
T
```

Explanation of Output for Sample Input 2
The first string is composed of all light letters.

The second string is composed of a heavy letter, then a light letter, and then a heavy letter.

La version française figure à la suite de la version anglaise.

Problem S3: Swipe

Problem Description
Swipe is a new mobile game that has recently exploded in popularity. In each level of Swipe, you are given 2 rows of integers that can be represented as arrays A and B of size N. The objective of Swipe is to beat each level by turning array A into array B.

There are two swipe operations you can perform on array A.

- Swipe right: Select the subarray $[\ell, r]$ and set $A_i = A_\ell$ for all $\ell \leq i \leq r$.
- Swipe left: Select the subarray $[\ell, r]$ and set $A_i = A_r$ for all $\ell \leq i \leq r$.

For example, starting with array $A = [0, 1, 2, 3, 4, 5]$, if we swipe right on $[2, 4]$, we would obtain the array $[0, 1, 2, 2, 2, 5]$. If instead, we started with the same array A, and swiped left on $[3, 5]$, we would obtain the array $[0, 1, 2, 5, 5, 5]$. Note that these arrays are 0-indexed.

Unfortunately, the game is bugged and contains levels that are impossible to beat. Determine if it is possible to transform array A into array B. If it is possible, determine a sequence of swipe operations that transforms array A into array B.

Input Specification
The first line of input will consist of one positive integer N, representing the length of each of the two arrays of integers.

The second line of input contains N space separated integers contained in array A.

The third line of input contains N space separated integers contained in array B.

The following table shows how the available 15 marks are distributed:

Marks	**Bounds on N**	**Bounds on A_i and B_i**
2	$N = 2$	$1 \leq A_i, B_i \leq 3$
4	$1 \leq N \leq 8$	$1 \leq A_i, B_i \leq 8$
4	$1 \leq N \leq 500$	$1 \leq A_i, B_i \leq 3000$
5	$1 \leq N \leq 300\,000$	$1 \leq A_i, B_i \leq 300\,000$

Note that for a subtask worth M marks, you will receive $\left\lfloor \frac{M}{2} \right\rfloor$ marks for a solution that only correctly outputs the first line of output.

Output Specification
The first line of output will contain `YES` if there is a sequence of swipes that can transform array A into array B; otherwise, the first line of output will contain `NO`.

La version française figure à la suite de la version anglaise.

If the first line of output is `YES`, the next line contains a non-negative integer K ($K \leq N$), indicating the number of swipes.

Each of the next K lines contain three space-separated values: D_j, ℓ_j, and r_j. The value D_j will be either `R` or `L`, indicating that the jth swipe is either a right or left swipe, respectively. The values ℓ_j and r_j indicate the left-end and right-end of the swipe where $0 \leq \ell_j \leq r_j < N$.

Sample Input 1

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

Output for Sample Input 1

```
YES
1
R 1 2
```

Sample Input 2

```
4
1 2 4 3
1 4 2 3
```

Output for Sample Input 2

```
NO
```

Sample Input 3

```
4
2 1 4 3
2 1 4 3
```

Output for Sample Input 3

```
YES
0
```

La version française figure à la suite de la version anglaise.

Problem S4: Painting Roads

Problem Description
Alanna, the mayor of Kitchener, has successfully improved the city's road plan. However, a travelling salesperson from the city of RedBlue complained that the roads are not colourful enough. Alanna's second job is to paint some of the roads.

Kitchener's road plan can be represented as a collection of N intersections with M roads, where the i-th road connects intersections u_i and v_i. All roads are initially grey. Alanna would like to paint some of the roads in red or blue such that the following condition is satisfied:

- Whenever there is a grey road that connects u_i and v_i, there is also a path of roads from u_i to v_i such that the roads on the path alternate between red and blue, without any of the roads on this path being grey.

To lower the city's annual spending, Alanna would like to minimize the number of painted roads. Can you help Alanna design a plan that meets all the requirements?

Input Specification
The first line contains two integers N and M ($1 \leq N, M \leq 2 \cdot 10^5$).

The i-th of the next M lines contains two integers u_i and v_i, meaning that there exists a road from intersection u_i to intersection v_i ($1 \leq u_i, v_i \leq N, u_i \neq v_i$).

There is at most one road between any unordered pair of intersections.

The following table shows how the available 15 marks are distributed:

Marks	Additional Constraints
2	There is a road connecting intersection i with intersection $i+1$ for all $1 \leq i < N$ (and possibly other roads).
3	We can reach any intersection from any other intersection, and $N = M$.
3	No road belongs to two or more simple cycles (see Definition below).
7	None

Definition: if we denote a road between intersections u and v as $u \leftrightarrow v$, then a *simple cycle* is a sequence $w_1 \leftrightarrow w_2 \leftrightarrow \ldots \leftrightarrow w_k \leftrightarrow w_1$ where $k \geq 3$ and all w_i are distinct.

Output Specification
Output a string of M characters, representing the paint plan. The i-th character should be `R` if the i-th road is to be painted red, `B` if i-th road is to be painted blue, or `G` (for "grey") if the i-th road is to be left unpainted.

La version française figure à la suite de la version anglaise.

Remember that you must minimize the number of painted roads while satisfying the condition. If there are multiple possible such plans, output any of them.

Sample Input 1

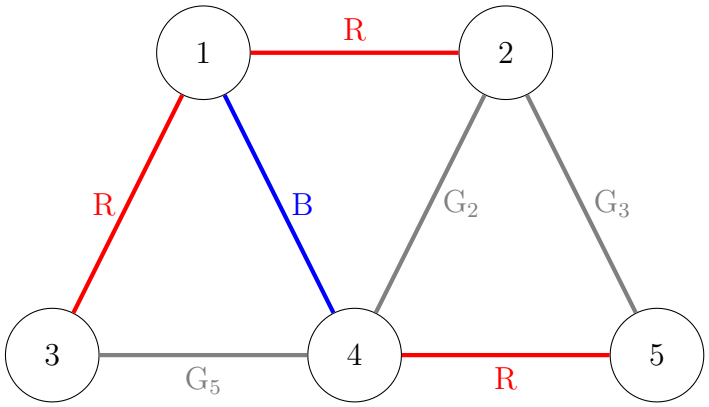
```
5 7
1 2
2 4
5 2
4 5
4 3
1 3
1 4
```

Output for Sample Input 1

```
RGGRGRB
```

Explanation of Output for Sample Input 1

A diagram of the intersections along with a valid paint plan that minimizes the number of painted roads is shown below. Note that the colours are shown on each road as R (red), B (blue), or G (grey).



All the unpainted roads satisfy the condition:

- The 2nd road, labelled G_2, connects intersection 2 with intersection 4. The path through intersections $2, 1, 4$ alternates red, blue.
- The 3rd road, labelled G_3, connects intersection 5 with intersection 2. The path through intersections $5, 4, 1, 2$ alternates red, blue, red.
- The 5th road, labelled G_5, connects intersection 4 with intersection 3. The path through intersections $4, 1, 3$ alternates blue, red.

La version française figure à la suite de la version anglaise.

Sample Input 2

```
4 2
1 2
3 4
```

Output for Sample Input 2

```
BB
```

Explanation of Output for Sample Input 2

Note that it is possible for Kitchener to be disconnected.

La version française figure à la suite de la version anglaise.

Problem S5: Chocolate Bar Partition

Problem Description
Maxwell has a chocolate bar that he wants to share with his friends. The chocolate bar can be represented as a 2 by N array of integers $T_{i,j}$, the tastiness of each square. Maxwell would like to split the entire chocolate bar into connected parts such that the average (mean) tastiness of the chocolate bar is the same for each part. Maxwell would like to know what is the maximum number of connected parts he can split his chocolate bar into as described above.

A part is considered connected if you can visit every cell by moving up, down, left or right.

Input Specification
The first line of input will consist of one positive integer N, representing the length of the chocolate bar.

The second line of input contains N spaced integers representing the top row of the chocolate bar where the j-th integer from the left represents $T_{1,j}$.

Similarly, the third line of input contains N spaced integers representing the bottom row of the chocolate bar where the j-th integer from the left represents $T_{2,j}$.

The following table shows how the available 15 marks are distributed:

Marks	**Bounds on N**	**Bounds on $T_{i,j}$**
2	$N = 2$	$0 \leq T_{i,j} \leq 5$
2	$1 \leq N \leq 8$	$0 \leq T_{i,j} \leq 20$
1	$1 \leq N \leq 20$	$0 \leq T_{i,j} \leq 20$
2	$1 \leq N \leq 100$	$0 \leq T_{i,j} \leq 20$
2	$1 \leq N \leq 1\,000$	$0 \leq T_{i,j} \leq 100$
3	$1 \leq N \leq 2\,000$	$0 \leq T_{i,j} \leq 100\,000$
3	$1 \leq N \leq 200\,000$	$0 \leq T_{i,j} \leq 100\,000\,000$

Output Specification
Output a single integer, representing the maximum number of connected parts Maxwell can split his chocolate bar into.

Sample Input 1
```
2
5 4
6 5
```

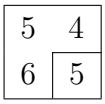
La version française figure à la suite de la version anglaise.

Output for Sample Input 1

```
2
```

Explanation of Output for Sample Input 1

An example of how to split this chocolate bar optimally into 2 parts is to have the bottom right corner as its own part and the rest of the chocolate as the second part, as shown below.



Each piece will have an average tastiness of 5.

Sample Input 2

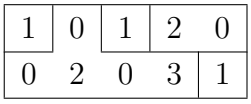
```
5
1 0 1 2 0
0 2 0 3 1
```

Output for Sample Input 2

```
5
```

Explanation of Output for Sample Input 2

One way to get the optimal split is shown in the following picture:



Note that each piece has an average tastiness of 1.

La version française figure à la suite de la version anglaise.

2024 Canadian Computing
Problem 1

Treasure Hunt

Time Limit: 4 seconds

Problem Description

Perry the Pirate is sailing the seven seas! He has a map consisting of N islands connected by a network of M sea routes. The i-th sea route connects islands a_i and b_i and costs c_i coins to traverse in either direction. As it turns out, fighting off sea monsters can be quite expensive. In search of his next big plunder, Perry has scouted out each of the N islands and has determined that the i-th island contains a treasure chest with v_i coins inside.

It remains for him to plan out his next journey. He decides that he will sail through some (possibly empty) path of sea routes starting at island x and ending at island y. At the end of his journey, he will open the chest at island y and collect his well-earned booty.

There is one small problem though: Perry doesn't know what island he's currently on! Thus, for every possible starting island x, he would like to know the maximum possible number of coins he can earn out of all journeys starting at island x. Can you help him compute these values? You may assume Perry has enough coins to traverse any path of sea routes he chooses; he only cares about the net profit of his next journey.

Input Specification

The first line of input contains two space-separated integers N and M.

The second line of input contains N space-separated integers $v_1, v_2, \ldots, v_N$ ($0 \leq v_i \leq 10^9$).

The next M lines each contain three space-separated integers a_i, b_i ($1 \leq a_i, b_i \leq N$), and c_i ($0 \leq c_i \leq 10^9$).

It is guaranteed that there is at most one sea route between any pair of islands and each sea route connects two distinct islands.

Marks Awarded	**Bounds on N**	**Bounds on M**	**Additional constraints**
5 marks	$1 \leq N \leq 3\,000$	$0 \leq M \leq 3\,000$	None
5 marks	$1 \leq N \leq 10^6$	$0 \leq M \leq 10^6$	For all i, either $c_i = 0$ or $c_i = 10^9$
7 marks	$1 \leq N \leq 10^6$	$0 \leq M \leq 10^6$	Exactly one path of sea routes between any pair of islands
8 marks	$1 \leq N \leq 10^6$	$0 \leq M \leq 10^6$	None

Output Specification

Output N lines, where the x-th line contains the maximum possible net profit (in coins) of any journey starting at island x.

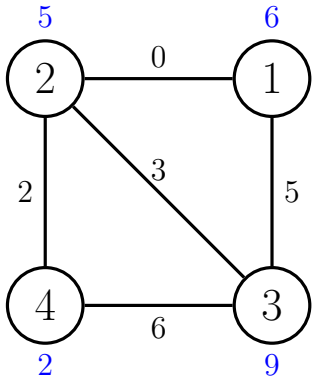
Sample Input

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

Output for Sample Input

```
6
6
9
4
```

Explanation of Output for Sample Input



For the first and third islands, it is best to just stay and open the chest on the island itself.

For the second island, Perry can travel to the first island and open the chest there. This has a net profit of $6 - 0 = 6$ coins and is the best possible net profit.

For the fourth island, Perry can to travel to the second and then the third island and open the chest there. This has a net profit of $9 - 2 - 3 = 4$ coins and is the best possible net profit.

2024 Canadian Computing
Problem 2

Pizza Party

Time Limit: 4 seconds

Problem Description

Troy is fleshing out the details of his latest initiative, HackCCO! Everyone knows that the biggest appeal of any hackathon is the free food. As such, to ensure the unparalleled success of HackCCO, Troy ordered a comically large cart stacked with N pizzas where the i-th pizza from the top of the cart has flavour a_i.

After the pizza cart arrives, Troy needs to arrange them into some number of stacks on a long table. To do this, he takes the pizzas one-by-one from the top of the cart and moves them onto the table, each time either placing the pizza on top of another stack of pizzas or forming a new stack on the table.

The N hungry HackCCO participants are lined up to get pizza from the table, one-by-one. Troy knows that the i-th person in line has a favourite pizza flavour of b_i. When the i-th person walks up to the table, if they see any pizzas of their favourite flavour at the top of any stack they will take any one of them at random. Otherwise, they won't take anything and will leave the table hungry.

Of course, hungry coders are not happy coders, so Troy doesn't want anyone to leave the table hungry. Thus, he asks you to help him find an arrangement of pizzas on the table such that it is possible for nobody to leave hungry. Furthermore, out of all such arrangements, Troy wants you to find one that creates the smallest number of stacks on the table (after all, tables can only get so long). Help him find such an arrangement or determine that it's impossible!

Input Specification

The first line of input contains a single integer N.

The second line of input contains N space-separated integers $a_1, \ldots, a_N$ ($1 \leq a_i \leq N$).

The third line of input contains N space-separated integers $b_1, \ldots, b_N$ ($1 \leq b_i \leq N$).

Marks Awarded	**Bounds on N**	**Additional constraints**
3 marks	$1 \leq N \leq 10^6$	$1 \leq a_i, b_i \leq 2$
4 marks	$1 \leq N \leq 5\,000$	All a_i are distinct
5 marks	$1 \leq N \leq 10^6$	All a_i are distinct
6 marks	$1 \leq N \leq 5\,000$	None
7 marks	$1 \leq N \leq 10^6$	None

[Post-CCO edit: Subtasks 4 and 5 may not be solvable efficiently. CCO competitors were judged only on subtasks 1-3.]

Output Specification

If it is impossible to arrange the pizzas as desired, output `-1`.

Otherwise, your output should consist of three lines. On the first line output K, the minimum number of stacks required. On the second line output N space-separated integers $c_1, \ldots, c_N$ ($1 \leq c_i \leq K$), indicating that the i-th pizza should be placed on stack c_i. On the third line output N space-separated integers $d_1, \ldots, d_N$ ($1 \leq d_i \leq K$), indicating that the i-th person in line takes their pizza from the d_i-th stack. This stack must have a pizza of flavour b_i at the top when the i-th participant walks up to get their pizza.

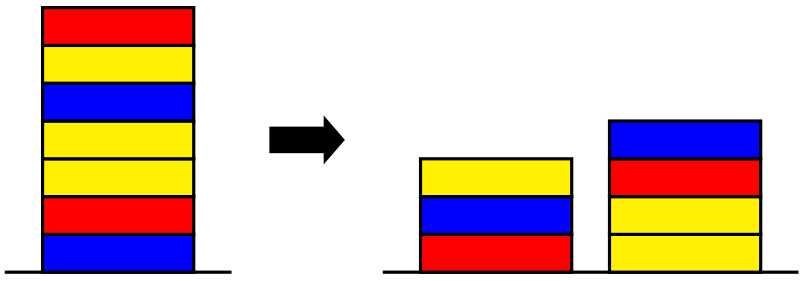
Sample Input 1

```
7
1 2 3 2 2 1 3
2 3 1 2 3 2 1
```

Output for Sample Input 1

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

Explanation of Output for Sample Input 1



An illustration of the arrangement of pizzas on the table is shown above where red, yellow, and blue boxes represent pizzas of flavours 1, 2, and 3 respectively.

Sample Input 2

```
2
1 2
1 1
```

Output for Sample Input 2

```
-1
```

2024 Canadian Computing
Problem 3

Summer Driving

Time Limit: 6 seconds

Problem Description

In Ontario, there are N cities numbered from 1 to N. There are $N-1$ roads numbered from 1 to $N-1$, where the i-th road connects city u_i and city v_i. It is possible to travel from any city to any other city using these roads.

Alice and Bob are travelling together, starting at city R. To make their driving experience more interesting, they devise the following game.

Alice and Bob will alternate turns, starting with Alice. On Alice's turn, she must drive along **exactly** A distinct roads that they have never traversed before in either direction. On Bob's turn, he must drive along **up to** B distinct roads (possibly zero), but some of these roads may have been traversed before.

Eventually, it will be Alice's turn, but it will be impossible for her to drive along exactly A distinct roads that they have never used before. When this happens, the game enters a final phase before Alice does any more driving. In this final phase, Bob drives along **up to** B distinct roads (possibly zero) that they have **never traversed before** in either direction.

Alice wants to end up in a city with as large a number as possible, while Bob wants to end up in a city with a small number. What is the city that Alice and Bob end their journey in when they both play optimally?

Input Specification

The first line of input contains four space-separated integers, N, R, A, and B ($1 \leq R, A, B \leq N$).

The next $N-1$ lines of input each contain two space-separated integers u_i and v_i ($1 \leq u_i, v_i \leq N, u_i \neq v_i$), describing a road.

Marks Awarded	Bounds on N	Additional Constraints
1 mark	$2 \leq N \leq 300\,000$	$A \leq B$
4 marks	$2 \leq N \leq 300\,000$	There are at most two roads connected to any city, and at most one road connected to city R.
5 marks	$2 \leq N \leq 300$	No additional constraints.
3 marks	$2 \leq N \leq 3\,000$	No additional constraints.
4 marks	$2 \leq N \leq 100\,000$	$B \leq 10$
6 marks	$2 \leq N \leq 100\,000$	No additional constraints.
2 marks	$2 \leq N \leq 300\,000$	No additional constraints.

Output Specification
Output the city that Alice and Bob end their journey in, assuming they both play optimally.

Sample Input 1
```
9 6 2 1
1 3
1 6
2 4
2 5
2 7
3 9
4 6
4 8
```

Output for Sample Input 1
```
2
```

Explanation of Output for Sample Input 1
On Alice's first turn, she has three options: Drive to city 2, 3, or 8.

If Alice drives to city 2, Bob can choose not to drive on any roads in his turn. Then, it will be impossible for Alice to drive along 2 distinct roads that were never traversed before, so the game enters the final phase. Bob can choose not to drive on any roads during this final phase, ending in city 2.

If Alice drives to city 3, Bob can choose to drive 1 road to city 1. Then, it will be impossible for Alice to drive along 2 distinct roads that were never traversed before, so the game enters the final phase. Bob's only option is to not drive on any roads during this final phase, ending in city 1.

If Alice drives to city 8, Bob can choose to drive 1 road to city 4. Then, Alice can drive to either city 5 or 7. In both cases, Bob can then drive 1 road to city 2. Alice can no longer drive along 2 distinct roads that were never traversed before, so the game enters the final phase. Bob can choose not to drive on any roads during this final phase, ending in city 2.

In all cases, Bob can force the game to end in city 1 or 2. Bob cannot force the game to end in city 1, so under optimal play, the game ends in city 2.

Sample Input 2
```
7 2 3 2
2 7
7 3
3 1
1 4
```

```
4 5
5 6
```

Output for Sample Input 2

```
3
```

2024 Canadian Computing
Problem 4

Infiltration

Time Limit: 1 second

Problem Description

Ondrej and Edward are spies and they are going to take down the evil organization AQT. To do so, they will need to infiltrate into the AQT base. The base can be modelled as a tree with $N = 100$ rooms, labelled from 0 to $N - 1$. Ondrej and Edward's plan to infiltrate the base is to first get kidnapped and then meet up together before executing their plan. When kidnapped, the two will be placed into different rooms unknown to each other. Once they are placed into the rooms, they will both break free at midnight and try to meet up with each other before executing their plan.

Their plan to meet up is as follows. At every odd minute, Ondrej can choose to stay at his current room or move to an adjacent room. At every even minute, Edward can choose to stay at his current room or move to an adjacent room.

A strategy is defined as the following. Let $V(A, R, T)$ denote the room agent A should be at assuming that they were at room R at midnight and it is currently T minutes after midnight. The strategy should match the conditions above. The agents are said to meet up at time $t(o, e)$, which is the first time where $V(\text{Ondrej}, o, t(o, e)) = V(\text{Edward}, e, t(o, e))$.

Ondrej and Edward want to meet up as fast as possible, relative to the distance between their two starting rooms. The distance $d(o, e)$ is the minimum number of corridors that must be traversed to reach o from e. Please help find a strategy that minimizes the maximum $\frac{t(o,e)}{d(o,e)}$ across all pairs of different rooms o and e.

Input Specification

The first line of input will contain N ($N = 100$). [Post-CCO edit: If the value of N is anything other than 100, exit the program immediately.]

The next $N - 1$ lines will each contain two space-separated integers, denoting the labels of two rooms with a bidirectional corridor between them.

Output Specification

First output a positive number T, the number of entries per starting room. Note that $T \leq 1440$ must be satisfied, otherwise you will be awarded no points.

Then, output Ondrej's strategy, followed by Edward's strategy.

To output an agent's strategy, output N lines, where the n-th line (starting from 0) represents the agent's path if they start at room n. For each line, output T spaced integers: The room

label that the agent should be in at time $1, 2, \ldots, T$.

Scoring

If the output is invalid or there exists a test case and a pair of different rooms o and e where the agents do not meet at or before time T, then no points will be awarded.

Otherwise, let S be the maximum among all test cases and pairs of o and e ($o \neq e$) of the value of $\frac{t(o,e)}{d(o,e)}$. The following table shows how the available 25 marks are distributed:

Score	**Bounds on S**
3	$200 < S \leq 1440$
6	$100 < S \leq 200$
8	$50 < S \leq 100$
10	$40 < S \leq 50$
12	$30 < S \leq 40$
15	$25 < S \leq 30$
17	$20 < S \leq 25$
18	$19 < S \leq 20$
19	$18 < S \leq 19$
20	$17 < S \leq 18$
21	$16 < S \leq 17$
22	$15 < S \leq 16$
25	$S \leq 15$

Sample Input 1

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

Output for Sample Input 1

```
8
2 2 4 4 1 1 1 1
1 1 1 1 1 1 1 1
3 3 2 2 3 3 2 2
3 3 2 2 0 0 2 2
4 4 4 4 2 2 2 2
0 2 2 3 3 3 3 2
```

```
1 4 4 2 2 0 0 0
2 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3
4 1 1 1 1 4 4 4
```

Explanation of Output for Sample Input 1

Note that this is an invalid test case as $N \neq 100$, so it will not appear in the test cases when judging. The output for the test case is valid. Note that this would not score any points because if Ondrej starts at room 1 and Edward starts at room 3, then they will never meet each other.

2024 Canadian Computing

PROBLEM 5

Heavy Light Decomposition

Time Limit: 4 seconds

Problem Description

In an array containing only positive integers, we say an integer is heavy if it appears more than once in the array, and light otherwise.

An array is good if the integers in the array alternate between light and heavy.

Given an array $a_1, \ldots, a_N$, count the number of ways to partition it into some number of contiguous subarrays such that each subarray, when considered as an array on its own, is good. As the answer may be large, output it modulo 1 000 003.

Input Specification

The first line of input contains a single integer, N.

The next line contains N integers $a_1, \ldots, a_N (1 \leq a_i \leq N)$.

Marks Awarded	Bounds on N	Additional Constraints
3 marks	$2 \leq N \leq 50\,000$	For each i, $a_i \leq 26$.
4 marks	$2 \leq N \leq 5\,000$	No additional constraints.
5 marks	$2 \leq N \leq 500\,000$	If i is odd, then $a_i = 1$.
6 marks	$2 \leq N \leq 500\,000$	Any number appears at most twice in the array.
7 marks	$2 \leq N \leq 500\,000$	No additional constraints.

Output Specification

The number of ways to partition the array into good contiguous subarrays, modulo 1 000 003.

Sample Input 1

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

Output for Sample Input 1

```
4
```

Explanation of Output for Sample Input 1

There are four valid partitions of [1, 2, 3, 2, 3]:

- [1], [2], [3], [2], [3]
- [1], [2, 3, 2], [3]
- [1], [2], [3, 2, 3]
- [1, 2, 3, 2], [3]

Sample Input 2

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

Output for Sample Input 2

```
6
```

2024 Canadian Computing

PROBLEM 6

Telephone Plans

Time Limit: 4 seconds

Problem Description

The "Dormi's Fone Service" is now the only telephone service provider in CCOland. There are N houses in CCOland, numbered from 1 to N. Each telephone line connects two distinct houses such that all the telephone lines that ever exist form a forest.

There is an issue where the phone lines are faulty, and each phone line only exists for a single interval of time. Two houses can call each other at a certain time if there is a path of phone lines that starts at one of the houses and ends in the other house at that time.

We would like to process Q queries of the following forms:

- `1 x y`: Add a phone line between houses x and y. It is guaranteed that a phone line between houses x and y was never added before.
- `2 x y`: Remove the phone line between houses x and y. It is guaranteed that a phone line currently exists between houses x and y.
- `3 t`: Compute the number of pairs of different houses that can call each other at some time between the current query and t queries ago. To be more clear, let G_q be the state of CCOland after the q-th query, where G_0 is the state of CCOland before any queries. If this is the s-th query, then count the number of pairs of houses that are connected in at least one of $G_{s-t}, G_{s-t+1}, \ldots, G_s$.

Also, some test cases may be encrypted. For the test cases that are encrypted, the arguments `x`, `y`, or `t` are given as the bitwise xor of the true argument and the answer to the last query of type `3` (if there have been no queries of type `3`, then the arguments are unchanged).

Input Specification

The first line of input will contain E ($E \in \{0, 1\}$). $E = 0$ denotes that the input is not encrypted, while $E = 1$ denotes that the input is encrypted.

The second line contains two space-separated integers N and Q, representing the number of houses in CCOland and the number of queries, respectively.

The next Q lines contain queries as specified above (queries are encrypted or not depending on E).

For the q-th query ($1 \leq q \leq N$), it is guaranteed that (after decrypting if $E = 1$) $1 \leq x, y \leq N$ for type `1` and `2` queries and $0 \leq t \leq q$ for type `3` queries.

Marks Awarded	Bounds on N	Bounds on Q	Encrypted?
3 marks	$1 \leq N \leq 30$	$1 \leq Q \leq 150$	$E = 0$
2 marks	$1 \leq N \leq 30$	$1 \leq Q \leq 150$	$E = 1$
4 marks	$1 \leq N \leq 2\,000$	$1 \leq Q \leq 6\,000$	$E = 0$
2 marks	$1 \leq N \leq 2\,000$	$1 \leq Q \leq 6\,000$	$E = 1$
4 marks	$1 \leq N \leq 100\,000$	$1 \leq Q \leq 300\,000$	$E = 0$
5 marks	$1 \leq N \leq 100\,000$	$1 \leq Q \leq 300\,000$	$E = 1$
5 marks	$1 \leq N \leq 500\,000$	$1 \leq Q \leq 1\,500\,000$	$E = 1$

Output Specification

For each query of type `3`, output the answer to the query on a new line.

Sample Input 1

```
0
4 12
3 0
1 1 2
3 0
1 1 3
3 0
3 5
2 2 1
3 0
3 8
1 1 4
3 0
3 11
```

Output for Sample Input 1

```
0
1
3
3
1
3
3
5
```

Explanation of Output for Sample Input 1

This test case is not encrypted.

For the 1st query, no pairs of different houses could have called each other.

For the 3rd query, only houses 1 and 2 could have called each other.

For the 5th query, $\{(1,2),(1,3),(2,3)\}$ is the set of pairs that could have called each other. The 6th query is the same.

For the 8th query, only houses 1 and 3 could have called each other.

For the 9th query, there is a point in time where $\{(1,2),(1,3),(2,3)\}$ could have called each other.

For the 11th query, the set of pairs that could have called each other is $\{(1,3),(1,4),(3,4)\}$.

For the 12th query, the set of pairs that could have called each other at any previous time is $\{(1,2),(1,3),(1,4),(2,3),(3,4)\}$.

Sample Input 2

```
1
4 12
3 0
1 1 2
3 0
1 0 2
3 1
3 6
2 1 2
3 3
3 9
1 2 7
3 3
3 8
```

Output for Sample Input 2

```
0
1
3
3
1
3
3
5
```

Explanation of Output for Sample Input 2

Encrypted version of sample 1.

J1 Conveyor Belt Sushi

The first problem on this year's contest involves printing out the result of an arithmetic expression. The expression consists of three input values representing the number of plates of each colour, and constant values which are the cost of each colour of plate.

J2 Dusa And The Yobis

This year, a twist was introduced at this early point of the contest. We are told Dusa's starting size and the size of the Yobis, but we are not told how many Yobis there are. This means that a while loop, or the equivalent, is needed instead of a for loop which might be more familiar. That is, we can continue reading input and adding the size of the current Yobi to the size of the Dusa while the size of the Dusa is less than the size of the current Yobi. Put another way, the loop continues until a Yobi is encountered that is the same size as the Dusa or larger.

J3 Bronze Count

There are several different ways of solving this problem.

One approach is to maintain six variables storing values for the gold, silver and bronze score as well as a count of the number of participants with each of these scores. Taking care to initialize these variables properly, they can then be updated as needed given each new score. This approach can be used to pass all three subtasks.

Another approach is to first store a collection of all the scores in a data structure (e.g. in a list in Python or an ArrayList in Java). Then the maximum score (gold score) can be calculated and all occurrences of this score removed from this collection. Then the new maximum score (silver score) can be calculated and all occurrences of this score removed from the updated smaller collection. The maximum of the scores remaining in this even smaller new collection is the bronze score and the number of times it occurs can be computed and printed. This approach will pass the first two subtasks. Whether or not this approach passes the last subtask depends on how computing and removing occurrences of the maximum score is implemented. As long as this is done by passing over the collection a fixed/constant number of times (i.e. increasing the size of the collection does not affect how often a pass is made over the collection), this approach should also pass the third subtask.

Sorting can also be used here. After sorting the scores, the bronze score can be identified quickly for the first subtask where the scores are distinct. Otherwise, a loop is needed to scan from highest to lowest score effectively finding the "boundaries" between the gold and silver scores and between the silver and bronze scores. Any built-in sorting function/method should be fast enough to pass the last subtask. It is also possible but challenging to implement a sorting algorithm which will be fast enough to earn full marks with

this strategy.

J4 Troublesome Keys

It can take some time to wrap your head around the behaviour of Alex's strange keyboard. To help us discuss it here, we will refer to the following five values.

pressed – the first line of input representing the keys pressed by Alex

displayed – the second line of input representing the letters displayed on the screen

silly – the letter corresponding to the silly key

wrong – the letter displayed when the silly key is pressed

quiet – the letter corresponding to the quiet key

For the first subtask, the quiet key is not pressed which means that the letter in pressed that is not in displayed must be silly. Also, wrong must be the letter that is in displayed but not in pressed. Finding a character that appears in one string but not the other can be done with a loop and search. The search can be accomplished with a built-in function or nested second loop. (In general, we can check whether or not the quiet key is pressed by comparing the lengths of pressed and displayed.)

For the other subtasks, we can still begin by determining wrong as above. Similarly, we can determine a pair of letters x and y that correspond to silly and quiet, but we need to work a bit harder to determine whether x corresponds to silly and y corresponds to quiet, or the other way around. For the second subtask, we can use the fact that the first troublesome key pressed is the silly key. Otherwise, one way to do this is to simulate Alex's keyboard by replacing all occurrences of x in pressed with wrong and removing y from pressed. It the result equals displayed then we know that x corresponds to silly and y corresponds to quiet. Otherwise, we know that it is the other way around. To solve the final subtask where a large number of keys are pressed, the number of times we pass over the input strings cannot depend on N, the bound on their lengths. This can be achieved by storing the letters in pressed and displayed in a data structure that allows for efficient searches (e.g. a dictionary in Python or HashMap in Java.)

This problem can also be solved without predetermining wrong and candidates for silly and quiet. We can scan pressed and displayed left to right looking for the first place where they differ. For the second subtask, we then immediately know that silly must be the mismatched character in pressed and wrong must be the mismatched character in displayed. To determine quiet (if it was pressed), we can continue scanning until we find a mismatch that does not involve silly. We omit the details here, but for the third and fourth subtasks, we can extend this approach by considering cases and simulating the behaviour of Alex's keyboard once we determine silly or quiet.

J5 Harvest Waterloo

For the final problem in this year's competition, the fundamental task is to determine all the locations that the farmer can reach. By "visiting" each of these locations and summing the values of the pumpkins at these locations, we can produce the total value harvested by the farmer. This requires us to store the pumpkin patch in memory which must ultimately be done with a two-dimensional structure (e.g. a list of strings or two-dimensional array).

For the first subtask, the farmer can reach every location. Visiting every location can be done with a nested loop.

For the second subtask, we can begin at the farmer's starting location and move in one direction until blocked by hay or the edge of the patch. Since we know that the locations the farmer can reach form a

La version fran¸caisefigure`alasuitedelaversionanglai

rectangle, doing this up, down, left and right, allows us to determine the dimensions of the rectangle and the location of a corner of the rectangle. This allows us to use a nested loop as in the first subtask to visit all the locations that the farmer can reach.

A different strategy is required for the final two subtasks. We can maintain a collection of locations to visit. Initially this collection will consist only of the starting location of the farmer. Then we can repeatedly remove a location from the collection and if it has not yet been accounted for, add the value of the pumpkin at this location to a running total. Also, since the farmer can only move in four possible directions, every location we visit presents up to four potential new locations to add to the collection. Locations should not be added if they are not reachable (contain hay or do not exist because we are at the edge of the patch) or if they have already been accounted for. One clever way to record that a location has been accounted for is to change the S, M or L at that location to a *. This general technique is an example of what is called a flood fill algorithm.

Notice that the collection of locations to visit can theoretically grow very large and contain many duplicates. This is because, for example, visiting the location at row 4 and column 7 could result in us adding the location at row 3 and column 7 to the collection, but this same location might be added when visiting row 3 and column 8 (or row 2 and column 7, or row 3 and column 6). Avoiding this is needed to earn 15 marks.

Depending on how the collection is implemented, the full solution described here is called breadth-first search (BFS) or depth-first search (DFS). DFS can also be implemented using a recursive function or method which is a function or method that calls itself. For recursive functions, the collection is implicit and stored in a portion of memory usually referred to as the call stack. Different languages and machines can put different limits on how large the call stack can be. Python tends to be especially restrictive and so using recursion and Python for this problem likely requires you to instruct the system to make enough room by importing

the sys module and issuing a command such as sys.setrecursionlimit(100000).

S1 Hat Circle

Subtask 1
We note that there can only be either 2 or 4 people at the table. This can be solved with conditional statements. In the case that there are only 2 people, they are seated across from each other, so we just compare the values of both hats. With 4 people, we can compare the equality of the hats at seat 1 and 3 and seat 2 and 4.
Subtask 2
All people have the same hat number 1. This means we can simply count the number of people at the table.
Subtask 3
Since people in even-numbered seats have hat number 1 and people in odd-numbered seats have hat number 0, they will only see matching hats if the number of people is divisible by 4.
Subtask 4
There was no specific intended approach for this subtask. We observe that in a circular arrangement, the $\frac{N}{2}$ person in seat i is directly opposite to the person in seat $i + \frac{N}{2}$.
One possible solution would be to store for each hat number, a list of the seat numbers of the people wearing that hat number. We can then iterate over the lists for each hat number and check if for each seat i in a $\frac{N}{2}$ list, if both i and $i + \frac{N}{2}$ exist in the same list.
Subtask 5
We notice that we can do the check directly on the input as a list. For each person at seat i, we check if the $\frac{N}{2}$ opposite person at seat $i + \frac{N}{2}$ has the same hat number. Since each seat is indexed from 1 to N, if $i > \frac{N}{2}$ then $i + \frac{N}{2}$ will be greater than N. To ensure that we stay within bounds, there are a couple approaches we can use:

- loop from 1 to N and compare i and $(i + \frac{N}{2})\%N$ to count person at seat i;
- loop from 1 to N and subtract N from $i + \frac{N}{2}$ if it is greater than N to count person at seat i;
- loop from 1 to $\frac{N}{2}$ and count for person at seat i and seat $i + \frac{N}{2}$.

S2 Heavy-Light Composition

Subtask 1
Hard-code every possible string containing “a” and “b” of length between 2 and 4 (there are 30 such strings) and output whether it alternates between heavy “a” and light “b” or heavy “b” and light “a”.
Subtask 2
There was no intended solution for this subtask. It was intended to allow for possibly slower full solutions.
Subtask 3
We consider two cases for if it starts with “a” or not:
- Check that every second letter is “a” and that the rest are not “a”.
- Check that every second letter is not “a” and that the rest are “a”.

If either case is true, output T; otherwise, output F.
Subtask 4
First, create an array that will store the frequency of each letter in the string. Each element represents the count of a particular letter. Then, iterate over all letters in the string and increment the frequency of it in the array.
We realize we can extend the logic from subtask 3 with this array. If a letter has a frequency greater than 1 in the array, it is heavy; if not, it is light.
We consider two cases for if it starts with a heavy letter or not:
- Check that every second letter has a frequency greater than 1 and that the rest are light.
- Check that every second letter does not have a frequency greater than 1 and that the rest are heavy.

If either case is true, output T; otherwise, output F.

S3 Swipe

Subtask 1
Since N = 2, we can use casework on all possible cases. Alternatively, note that the only moves that can be made are to do nothing, swipe left on [0, 1], or swipe right on [0, 1]. After 1 swipe, additional swipes will

not change the array A, as both elements of A will be the same. Thus, we can try all possible moves, and check if array A becomes B. Otherwise, the answer will be NO.

Subtask 2

The intended solution for this subtask is to iteratively brute force all possible arrays by trying all possible swipes at each step. This can be implemented similarly to BFS, with a visited map, to ensure we don't visit the same array A twice. If we are able to reach array B, then we print out the sequence of swipes and return. As $N \leq 8$, with careful implementation, this should pass comfortably in time.

Subtask 3 and 4

La version fran¸caisefigure`alasuitedelaversionanglaise.

🔎

Let B be the "compressed" version of B, where all adjacent equal elements of B are removed. Then, the

🔎

answer is YES if and only if B is a sub-sequence of A.

To see why this is the case, notice that intersecting a necessary left swipe with a necessary right swipe will overwrite valuable elements. Take A = [1, 2] and B = [2, 1] as an example. In order to make the first element a 2, we must swipe left and we get A = [2, 2]. However, we cannot make the second element a 1 now, as the 1 has been overwritten. A similar argument follows if we first try to swipe right. Thus, it is impossible

🔎

for array A to turn into B. When B is not a sub-sequence of A, there will inevitably be a case where a necessary left swipe intersects a necessary right swipe, which makes it impossible.

To construct the solution, we employ a 2-pointers approach. Iterate i through each element of A, and increment j while A_i is equal to B_j. This will capture the range of elements that we need to swipe across. We push all left swipes into a list of swipes lef t, and right swipes into a list of swipes right. Notice that left swipes should be performed in increasing order of right endpoints, while right swipes should be performed in decreasing order of left endpoints, so that valuable elements will not get overwritten. Thus, we can push the reversed right into lef t to get the correct order of swipes.

This solution runs in $O(N)$ time, which is sufficient for full marks. Subtask 3 was meant for suboptimal implementations of the full solution.

S4 Painting Roads

Subtask 1

We can model the intersections and roads as an undirected graph. In this subtask, the roads connecting intersection i with intersection $i + 1$ for all $1 \leq i < N$ ensure that the graph is connected and has a Hamiltonian path $1 \leftrightarrow 2 \leftrightarrow \cdots \leftrightarrow N$.

First, observe that Alanna must paint at least $N - 1$ roads. If not, then the coloured roads will not form a connected subgraph. Because the original graph is connected, there must be a grey road connecting two different components of coloured roads, and there is no coloured path connecting the endpoints of this grey road.

Because we're guaranteed a Hamiltonian path $1 \leftrightarrow 2 \leftrightarrow \cdots \leftrightarrow N$, let's consider choosing these $N - 1$ roads to paint. Thinking about what constraints on the colours are necessary to satisfy the condition in the problem, we can realize that it suffices to colour the roads alternating between red and blue along the path $1 \leftrightarrow 2 \leftrightarrow \cdots \leftrightarrow N$.

Subtask 2

In this subtask, the graph is connected and has an equal number of edges and nodes. In other words, the graph forms a pseudotree. In particular, there is exactly one simple cycle in the graph.

Just like in subtask 1, it can be proven that Alanna must paint at least $N - 1$ roads. To achieve this bound, we can colour the edges of the cycle by alternating between red and blue, leaving exactly one cycle edge grey. The other edges should be coloured red or blue arbitrarily.

This cycle may be found by a carefully implemented graph-traversal algorithm like BFS or DFS.

Subtask 3

In this subtask, the graph is no longer guaranteed to be connected. The condition of no road belonging to two or more simple cycles can be rephrased as each connected component of the graph being a cactus graph.

La version fran¸caisefigure`alasuitedelaversionanglaise.

By generalizing the ideas from subtasks 1 and 2, the coloured edges should form a spanning forest of the graph. That would ensure that the coloured edges form the same connected components as the original graph while colouring the minimum number of edges to meet this requirement.

Let's consider choosing an arbitrary spanning forest of the graph to colour with red or blue. For each grey edge $u \leftrightarrow v$ not in the spanning forest, we can colour the path (in the corresponding tree) connecting u and v by alternating between red and blue. Because the component is a cactus, these paths do not share any edges, so we don't have to worry about these edges being involved in conflicting constraints.

An arbitrary spanning forest can be found with algorithms like BFS or DFS. To determine the correct $N-1$ colours, we cannot afford to spend $O(N)$ time per path because there could be as many as $\frac{N-1}{2}$ grey edges.

Instead, we need to identify the corresponding path in O(length of path) time. One way to do this is by first precomputing parents and depths of each node within a tree. Then, for each grey edge $u \leftrightarrow v$, trace out the path by repeatedly moving the deeper node (u or v) up to its parent until u and v coincide.

Subtask 4

The solution from subtask 3 no longer works because the paths induced by each grey edge are no longer disjoint, so it is possible that they cause some conflicting constraints on the colours. It is no longer enough to take an arbitrary spanning forest.
Guided by these intuitions, let's think about what conditions on the spanning forest would be nice to have. Let's focus on a single connected component, where the coloured edges form a tree, and let's root the tree arbitrarily. Consider a grey edge u ↔ v, and let the LCA (lowest common ancestor) of u and v in the tree be l. The constraints on the colours that this grey edge places are that (1) all edges between u and l must be different from its parent edge, (2) all edges between v and l must be different from its parent edge, and (3) if u ⌕ = l and v ⌕ = l, then two child edges of l are different.
Intuitively, most constraints are from types (1) and (2). It is actually possible to satisfy all type (1) and (2) constraints by colouring each edge by the parity of its depth in the tree, but this fails to satisfy any type (3) constraints. This motivates us to consider: Is there a spanning forest where we end up with no type (3) constraints?
In other words, we want a spanning forest with no cross edges : edges that do not connect an ancestor with a descendent. For undirected graphs, the DFS tree (tree formed by a depth-first traversal) has exactly this property we're looking for. This results in a very short solution to the full problem: For each connected component, take a DFS tree and colour the tree edges red or blue based on the parity of its depth.

S5 Chocolate Bar Partition

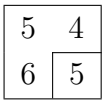
The first section will go over the full solution and the last paragraph will touch upon the intended solutions for some subtasks.
Let the mean of the entire array be µ. Notice that the mean of each component of a valid partition must be µ. Now consider the transformation of Ai,j = Ti,j − µ. This reduces the problem to splitting array A into the maximum number of parts such that each part has a sum of 0.
Now let Pj,i be the prefix sum array for Aj . That is,
⌕
i
Pj,i = Aj,c
c=1
La version fran¸caisefigure`alasuitedelaversionanglaise.
We will solve the new problem with dynamic programming. Let dp[k][i] represent the maximum number of parts we can split the first i columns of the array such that the sum of each part is 0 and there is a (possibly empty) component sticking out of the k-th row (row 1 or row 2). Thus, for the base case we have that dp[k][0] = 0.
We can incrementally update dp[k][i] starting from the smallest i. First set dp[k][i] to be the maximum of
1. dp[k][i − 1] (Ak,i is sticking out)
2. dp[3 − k][i − 1] (Ak,i and A3−k,i are in one component)
3. dp[k][j] + 1 such that P3−k,j = P3−k,i and j < i (Add a "line component" from A3−k,j+1 to A3−k,i)
4. dp[3 − k][j] + 1 such that Pk,i + P3−k,j = 0 (Finish the component by cutting a chunk from the first i cells on row 3 − k and the first j cells on row k)
Then if P1,i + P2,i = 0, simultaneously update dp[1][i] and dp[2][i] as max(dp[0][i], dp[1][i]) + 1 to signify that we split between column i and i + 1 (or we reach the end of i = N).
The final answer is dp[1][N]. To help understand how this is sufficient, it is helpful to draw out the transitions with multiple cases and notice that the sum of the component that is sticking out for dp[k][i] is Pk,i.
This results in an O
2
(N) algorithm as seen in cases 3 and 4. We can make it run in sub-quadratic time by maintaining maps of Pk,i → max dp[k][i] and Pk,i → max dp[3 − k][i] as we compute dp[k][i].
For the first 2 subtasks, we can generate all the possible partitions by hand for Subtask 1 or by running a clever brute force for Subtask 2. The other subtasks are used to reward suboptimal dynamic programming solutions such as using the sum of the component as a state, using the prefix of the first row and the second row simultaneously as a state or with suboptimal transitions.

Output for Sample Input 1

```
2
```

Explanation of Output for Sample Input 1

An example of how to split this chocolate bar optimally into 2 parts is to have the bottom right corner as its own part and the rest of the chocolate as the second part, as shown below.



Each piece will have an average tastiness of 5.

Sample Input 2

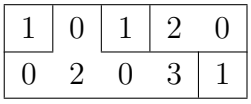
```
5
1 0 1 2 0
0 2 0 3 1
```

Output for Sample Input 2

```
5
```

Explanation of Output for Sample Input 2

One way to get the optimal split is shown in the following picture:



Note that each piece has an average tastiness of 1.

La version française figure à la suite de la version anglaise.