Codeforces Round #784 (Div. 4)

A. Division?

1 second, 256 megabytes

Codeforces separates its users into 4 divisions by their rating:

- For Division 1: $1900 \le \text{rating}$
- For Division 2: $1600 \le \text{rating} \le 1899$
- For Division 3: $1400 \le \text{rating} \le 1599$
- For Division 4: $\text{rating} \le 1399$

Given a rating, print in which division the rating belongs.

Input

The first line of the input contains an integer t ($1 \le t \le 10^4$) — the number of testcases.

The description of each test consists of one line containing one integer rating ($-5000 \le \text{rating} \le 5000$).

Output

For each test case, output a single line containing the correct division in the format "`Division X`", where X is an integer between 1 and 4 representing the division for the corresponding rating.

input
```
7
-789
1299
1300
1399
1400
1679
2300
```

output
```
Division 4
Division 4
Division 4
Division 4
Division 3
Division 2
Division 1
```

For test cases $1-4$, the corresponding ratings are -789, 1299, 1300, 1399, so all of them are in division 4.

For the fifth test case, the corresponding rating is 1400, so it is in division 3.

For the sixth test case, the corresponding rating is 1679, so it is in division 2.

For the seventh test case, the corresponding rating is 2300, so it is in division 1.

B. Triple

1 second, 256 megabytes

Given an array a of n elements, print any value that appears at least three times or print `-1` if there is no such value.

Input

The first line contains an integer t ($1 \le t \le 10^4$) — the number of test cases.

The first line of each test case contains an integer n ($1 \le n \le 2 \cdot 10^5$) — the length of the array.

The second line of each test case contains n integers $a_1, a_2, \dots, a_n$ ($1 \le a_i \le n$) — the elements of the array.

It is guaranteed that the sum of n over all test cases does not exceed $2 \cdot 10^5$.

Output

For each test case, print any value that appears at least three times or print `-1` if there is no such value.

input
```
7
1
1
3
2 2 2
7
2 2 3 3 4 2 2
8
1 4 3 4 3 2 4 1
9
1 1 1 2 2 2 3 3 3
5
1 5 2 4 3
4
4 4 4 4
```

output
```
-1
2
2
4
3
-1
4
```

In the first test case there is just a single element, so it can't occur at least three times and the answer is `-1`.

In the second test case, all three elements of the array are equal to 2, so 2 occurs three times, and so the answer is 2.

For the third test case, 2 occurs four times, so the answer is 2.

For the fourth test case, 4 occurs three times, so the answer is 4.

For the fifth test case, 1, 2 and 3 all occur at least three times, so they are all valid outputs.

For the sixth test case, all elements are distinct, so none of them occurs at least three times and the answer is `-1`.

C. Odd/Even Increments

1 second, 256 megabytes

Given an array $a = [a_1, a_2, \dots, a_n]$ of n positive integers, you can do operations of two types on it:

1. Add 1 to **every** element with an **odd** index. In other words change the array as follows: $a_1 := a_1 + 1, a_3 := a_3 + 1, a_5 := a_5 + 1, \dots$.
2. Add 1 to **every** element with an **even** index. In other words change the array as follows: $a_2 := a_2 + 1, a_4 := a_4 + 1, a_6 := a_6 + 1, \dots$.

Determine if after any number of operations it is possible to make the final array contain only even numbers or only odd numbers. In other words, determine if you can make all elements of the array have the same parity after any number of operations.

Note that you can do operations of both types any number of times (even none). Operations of different types can be performed a different number of times.

Input

The first line contains an integer t ($1 \le t \le 100$) — the number of test cases.

The first line of each test case contains an integer n ($2 \le n \le 50$) — the length of the array.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^3$) — the elements of the array.

Note that after the performed operations the elements in the array can become greater than 10^3.

Output

Output t lines, each of which contains the answer to the corresponding test case. As an answer, output "YES" if after any number of operations it is possible to make the final array contain only even numbers or only odd numbers, and "NO" otherwise.

You can output the answer in any case (for example, the strings "yEs", "yes", "Yes" and "YES" will be recognized as a positive answer).

input

```
4
3
1 2 1
4
2 2 2 3
4
2 2 2 2
5
1000 1 1000 1 1000
```

output

```
YES
NO
YES
YES
```

For the first test case, we can increment the elements with an even index, obtaining the array $[1, 3, 1]$, which contains only odd numbers, so the answer is "YES".

For the second test case, we can show that after performing any number of operations we won't be able to make all elements have the same parity, so the answer is "NO".

For the third test case, all elements already have the same parity so the answer is "YES".

For the fourth test case, we can perform one operation and increase all elements at odd positions by 1, thus obtaining the array $[1001, 1, 1001, 1, 1001]$, and all elements become odd so the answer is "YES".

D. Colorful Stamp

1 second, 256 megabytes

A row of n cells is given, all initially white. Using a stamp, you can stamp any two neighboring cells such that one becomes red and the other becomes blue. A stamp can be rotated, i.e. it can be used in both ways: as BR and as RB.

During use, the stamp must completely fit on the given n cells (it cannot be partially outside the cells). The stamp can be applied multiple times to the same cell. Each usage of the stamp recolors both cells that are under the stamp.

For example, one possible sequence of stamps to make the picture BRBBW could be WWWWW $\to$ WW<u>RB</u>W $\to$ <u>BR</u>RBW $\to$ B<u>RB</u>BW. Here W, R, and B represent a white, red, or blue cell, respectively, and the cells that the stamp is used on are marked with an underline.

Given a final picture, is it possible to make it using the stamp zero or more times?

Input

The first line contains an integer t ($1 \le t \le 10^4$) — the number of test cases.

The first line of each test case contains an integer n ($1 \le n \le 10^5$) — the length of the picture.

The second line of each test case contains a string s — the picture you need to make. It is guaranteed that the length of s is n and that s only consists of the characters W, R, and B, representing a white, red, or blue cell, respectively.

It is guaranteed that the sum of n over all test cases does not exceed 10^5.

Output

Output t lines, each of which contains the answer to the corresponding test case. As an answer, output "YES" if it possible to make the picture using the stamp zero or more times, and "NO" otherwise.

You can output the answer in any case (for example, the strings "yEs", "yes", "Yes" and "YES" will be recognized as a positive answer).

input

```
12
5
BRBBW
1
B
2
WB
2
RW
3
BRB
3
RBB
7
WWWWWWW
9
RBWBWRRBW
10
BRBRBRBRRB
12
BBBRWWRRRWBR
10
BRBRBRBRBW
5
RBWBW
```

output

```
YES
NO
NO
NO
YES
YES
YES
NO
YES
NO
YES
NO
```

The first test case is explained in the statement.

For the second, third, and fourth test cases, it is not possible to stamp a single cell, so the answer is "NO".

For the fifth test case, you can use the stamp as follows: WWW $\to$ W<u>RB</u> $\to$ <u>BR</u>B.

For the sixth test case, you can use the stamp as follows: WWW $\to$ W<u>RB</u> $\to$ <u>RB</u>B.

For the seventh test case, you don't need to use the stamp at all.

E. 2-Letter Strings

2 seconds, 256 megabytes

Given n strings, each of length 2, consisting of lowercase Latin alphabet letters **from 'a' to 'k'**, output the number of pairs of indices (i, j) such that $i < j$ and the i-th string and the j-th string differ in exactly one position.

In other words, count the number of pairs (i, j) $(i < j)$ such that the i-th string and the j-th string have **exactly** one position p $(1 \leq p \leq 2)$ such that $s_{ip} \neq s_{jp}$.

The answer may not fit into 32-bit integer type, so you should use 64-bit integers like `long long` in C++ to avoid integer overflow.

Input

The first line of the input contains a single integer t $(1 \leq t \leq 100)$ — the number of test cases. The description of test cases follows.

The first line of each test case contains a single integer n $(1 \leq n \leq 10^5)$ — the number of strings.

Then follows n lines, the i-th of which containing a single string s_i of length 2, consisting of lowercase Latin letters **from 'a' to 'k'**.

It is guaranteed that the sum of n over all test cases does not exceed 10^5.

Output

For each test case, print a single integer — the number of pairs (i, j) ($i < j$) such that the i-th string and the j-th string have **exactly** one position p $(1 \leq p \leq 2)$ such that $s_{ip} \neq s_{jp}$.

Please note, that the answer for some test cases won't fit into 32-bit integer type, so you should use at least 64-bit integer type in your programming language (like `long long` for C++).

input

```
4
6
ab
cb
db
aa
cc
ef
7
aa
bb
cc
ac
ca
bb
aa
4
kk
kk
ab
ab
5
jf
jf
jk
jk
jk
```

output

```
5
6
0
6
```

For the first test case the pairs that differ in exactly one position are: ("ab", "cb"), ("ab", "db"), ("ab", "aa"), ("cb", "db") and ("cb", "cc").

For the second test case the pairs that differ in exactly one position are: ("aa", "ac"), ("aa", "ca"), ("cc", "ac"), ("cc", "ca"), ("ac", "aa") and ("ca", "aa").

For the third test case, the are no pairs satisfying the conditions.

F. Eating Candies

1 second, 256 megabytes

There are n candies put from left to right on a table. The candies are numbered from left to right. The i-th candy has weight w_i. Alice and Bob eat candies.

Alice can eat any number of candies from the left (she can't skip candies, she eats them in a row).

Bob can eat any number of candies from the right (he can't skip candies, he eats them in a row).

Of course, if Alice ate a candy, Bob can't eat it (and vice versa).

They want to be fair. Their goal is to eat the same total weight of candies. What is the most number of candies they can eat in total?

Input

The first line contains an integer t $(1 \leq t \leq 10^4)$ — the number of test cases.

The first line of each test case contains an integer n ($1 \leq n \leq 2 \cdot 10^5$) — the number of candies on the table.

The second line of each test case contains n integers $w_1, w_2, \dots, w_n$ ($1 \leq w_i \leq 10^4$) — the weights of candies from left to right.

It is guaranteed that the sum of n over all test cases does not exceed $2 \cdot 10^5$.

Output

For each test case, print a single integer — the maximum number of candies Alice and Bob can eat in total while satisfying the condition.

input

```
4
3
10 20 10
6
2 1 4 2 4 1
5
1 2 4 8 16
9
7 3 20 5 15 1 11 8 10
```

output

```
2
6
0
7
```

For the first test case, Alice will eat one candy from the left and Bob will eat one candy from the right. There is no better way for them to eat the same total amount of weight. The answer is 2 because they eat two candies in total.

For the second test case, Alice will eat the first three candies from the left (with total weight 7) and Bob will eat the first three candies from the right (with total weight 7). They cannot eat more candies since all the candies have been eaten, so the answer is 6 (because they eat six candies in total).

For the third test case, there is no way Alice and Bob will eat the same non-zero weight so the answer is 0.

For the fourth test case, Alice will eat candies with weights $[7, 3, 20]$ and Bob will eat candies with weights $[10, 8, 11, 1]$, they each eat 30 weight. There is no better partition so the answer is 7.

G. Fall Down

1 second, 256 megabytes

There is a grid with n rows and m columns, and three types of cells:

- An empty cell, denoted with '`.`'.
- A stone, denoted with '`*`'.
- An obstacle, denoted with the lowercase Latin letter '`o`'.

All stones fall down until they meet the floor (the bottom row), an obstacle, or other stone which is already immovable. (In other words, all the stones just fall down as long as they can fall.)

Simulate the process. What does the resulting grid look like?

Input

The input consists of multiple test cases. The first line contains an integer t $(1 \le t \le 100)$ — the number of test cases. The description of the test cases follows.

The first line of each test case contains two integers n and m ($1 \le n, m \le 50$) — the number of rows and the number of columns in the grid, respectively.

Then n lines follow, each containing m characters. Each of these characters is either '.', '*', or 'o' — an empty cell, a stone, or an obstacle, respectively.

Output

For each test case, output a grid with n rows and m columns, showing the result of the process.

You don't need to output a new line after each test, it is in the samples just for clarity.

input

```
3
6 10
.*.*....*.
.*.......*
...o....o.
.*.*....*.
..........
.o......o*
2 9
...***ooo
.*o.*o.*o
5 5
*****
*....
*****
....*
*****
```

output

```
..........
...*....*.
.*.o....o.
.*........
.*......**
.o.*....o*

....**ooo
.*o**o.*o

.....
*...*
*****
*****
*****
```

H. Maximal AND

2 seconds, 256 megabytes

Let AND denote the bitwise AND operation, and OR denote the bitwise OR operation.

You are given an array a of length n and a non-negative integer k. You can perform **at most** k operations on the array of the following type:

- Select an index i $(1 \le i \le n)$ and replace a_i with a_i OR 2^j where j is any integer between 0 and 30 **inclusive**. In other words, in an operation you can choose an index i $(1 \le i \le n)$ and set the j-th bit of a_i to 1 $(0 \le j \le 30)$.

Output the maximum possible value of a_1 AND a_2 AND $\dots$ AND a_n after performing **at most** k operations.

Input

The first line of the input contains a single integer t $(1 \le t \le 100)$ — the number of test cases. The description of test cases follows.

The first line of each test case contains the integers n and k ($1 \le n \le 2 \cdot 10^5$, $0 \le k \le 10^9$).

Then a single line follows, containing n integers describing the arrays a ($0 \le a_i < 2^{31}$).

It is guaranteed that the sum of n over all test cases does not exceed $2 \cdot 10^5$.

Output

For each test case, output a single line containing the maximum possible AND value of a_1 AND a_2 AND $\dots$ AND a_n after performing **at most** k operations.

input

```
4
3 2
2 1 1
7 0
4 6 6 28 6 6 12
1 30
0
4 4
3 1 3 1
```

output

```
2
4
2147483646
1073741825
```

For the first test case, we can set the bit 1 (2^1) of the last 2 elements using the 2 operations, thus obtaining the array $[2, 3, 3]$, which has AND value equal to 2.

For the second test case, we can't perform any operations so the answer is just the AND of the whole array which is 4.