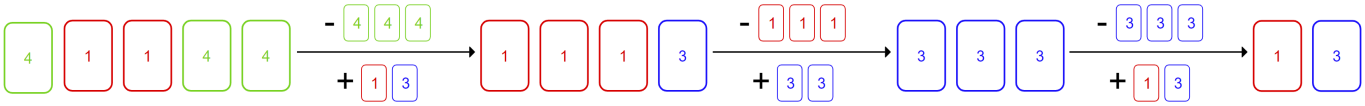
A. Card Exchange

Input file: standard input
Output file: standard output
Time limit: 1 second
Memory limit: 256 megabytes

You have a hand of n cards, where each card has a number written on it, and a fixed integer k. You can perform the following operation any number of times:

- Choose any k cards from your hand that all have the same number.
- Exchange these cards for $k - 1$ cards, each of which can have **any** number you choose (including the number written on the cards you just exchanged).

Here is one possible sequence of operations for the first example case, which has $k = 3$:



What is the minimum number of cards you can have in your hand at the end of this process?

Input

The first line of the input contains a single integer t ($1 \le t \le 500$) — the number of test cases. The description of the test cases follows.

The first line of each test case contains two integers n and k ($1 \le n \le 100, 2 \le k \le 100$) — the number of cards you have, and the number of cards you exchange during each operation, respectively.

The next line of each test case contains n integers $c_1, c_2, \ldots c_n$ ($1 \le c_i \le 100$) — the numbers written on your cards.

Output

For each test case, output a single integer — the minimum number of cards you can have left in your hand after any number of operations.

Standard Input	Standard Output
7 5 3 4 1 1 4 4 1 10 7 7 2 4 2 1 100 5 2 3 10 4 1 1 1 1 1 1 1 1 1 1 5 2 3 8 1 48 7 6 2 10 20 30 10 20 40 6 3 10 20 30 10 20 40	2 1 1 3 5 1 6

Note

The first example case corresponds to the picture above. The sequence of operations displayed there is optimal, so the answer is 2.

In the second example case, no operations can be performed, so the answer is 1.

In the fourth example case, you can repeatedly select 4 cards numbered with 1 and replace them with 3 cards numbered with 1, until there are 3 cards left.

,

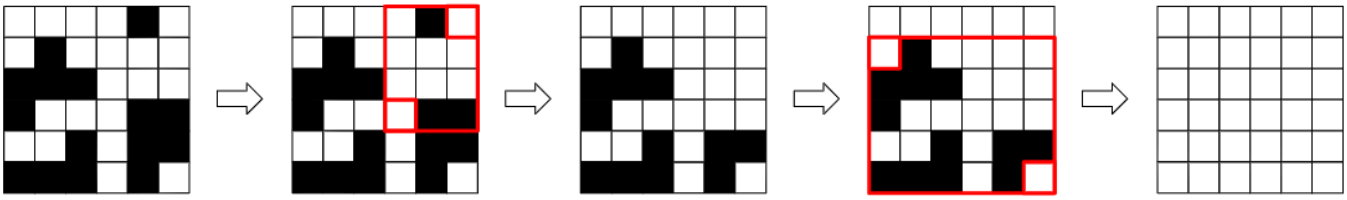
B. Rectangle Filling

Input file: standard input
Output file: standard output
Time limit: 1 second
Memory limit: 256 megabytes

There is an $n \times m$ grid of white and black squares. In one operation, you can select any two squares of the same color, and color all squares in the subrectangle between them that color.

Formally, if you select positions (x_1, y_1) and (x_2, y_2), both of which are currently the same color c, set the color of all (x, y) where $\min(x_1, x_2) \leq x \leq \max(x_1, x_2)$ and $\min(y_1, y_2) \leq y \leq \max(y_1, y_2)$ to c.

This diagram shows a sequence of two possible operations on a grid:



Is it possible for all squares in the grid to be the same color, after performing any number of operations (possibly zero)?

Input

The first line of the input contains a single integer t ($1 \leq t \leq 10^4$) — 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 ($1 \leq n, m \leq 500$) — the number of rows and columns in the grid, respectively.

Each of the next n lines contains m characters 'W' and 'B' — the initial colors of the squares of the grid.

It is guaranteed that the sum of $n \cdot m$ over all test cases does not exceed $3 \cdot 10^5$.

Output

For each test case, print "YES" if it is possible to make all squares in the grid the same color, and "NO" otherwise.

You can output the answer in any case (upper or lower). For example, the strings "yEs", "yes", "Yes", and "YES" will be recognized as positive responses.

Standard Input	Standard Output
8 2 1 W B 6 6 WWWWBW WBWWWW BBBWWW BWWWBB	NO YES YES YES YES NO YES NO

WWBWBB BBBWBW 1 1 W 2 2 BB BB 3 4 BWBW WBWB BWBW 4 2 BB BB WW WW 4 4 WWBW BBWB WWBB BBBB 1 5 WBBWB	

Note

In the first example, it is impossible to ever change the color of any square with an operation, so we output NO.

The second example is the case pictured above. As shown in that diagram, it is possible for all squares to be white after two operations, so we output YES.

In the third and fourth examples, all squares are already the same color, so we output YES.

In the fifth example we can do everything in two operations. First, select positions $(2, 1)$ and $(1, 4)$ and color all squares with $1 \leq x \leq 2$ and $1 \leq y \leq 4$ to white. Then, select positions $(2, 1)$ and $(3, 4)$ and color all squares with $2 \leq x \leq 3$ and $1 \leq y \leq 4$ to white. After these two operations all squares are white.

,

C. Everything Nim

Input file: standard input
Output file: standard output
Time limit: 2 seconds
Memory limit: 256 megabytes

Alice and Bob are playing a game on n piles of stones. On each player's turn, they select a positive integer k that is at most the size of the smallest **nonempty** pile and remove k stones from **each** nonempty pile at once. The first player who is unable to make a move (because all piles are empty) loses.

Given that Alice goes first, who will win the game if both players play optimally?

Input

The first line of the input contains a single integer t ($1 \le t \le 10^4$) — the number of test cases. The description of the test cases follows.

The first line of each test case contains a single integer n ($1 \le n \le 2 \cdot 10^5$) — the number of piles in the game.

The next line of each test case contains n integers $a_1, a_2, \ldots a_n$ ($1 \le a_i \le 10^9$), where a_i is the initial number of stones in the i-th pile.

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 line with the name of the winner, assuming both players play optimally. If Alice wins, print "Alice", otherwise print "Bob" (without quotes).

Standard Input	Standard Output
7 5 3 3 3 3 3 2 1 7 7 1 3 9 7 4 2 100 3 1 2 3 6 2 1 3 4 2 4 8 5 7 2 9 6 3 3 2 1 1000000000	Alice Bob Alice Alice Bob Alice Alice

Note

In the first test case, Alice can win by choosing $k = 3$ on her first turn, which will empty all of the piles at once.

In the second test case, Alice must choose $k = 1$ on her first turn since there is a pile of size 1, so Bob can win on the next turn by choosing $k = 6$.

,

D. Missing Subsequence Sum

```
Input file:     standard input
Output file:    standard output
Time limit:     2 seconds
Memory limit:   256 megabytes
```

You are given two integers n and k. Find a sequence a of non-negative integers of size at most 25 such that the following conditions hold.

- There is no subsequence of a with a sum of k.
- For all $1 \leq v \leq n$ where $v \neq k$, there is a subsequence of a with a sum of v.

A sequence b is a subsequence of a if b can be obtained from a by the deletion of several (possibly, zero or all) elements, without changing the order of the remaining elements. For example, $[5, 2, 3]$ is a subsequence of $[1, 5, 7, 8, 2, 4, 3]$.

It can be shown that under the given constraints, a solution always exists.

Input

The first line of the input contains a single integer t ($1 \leq t \leq 1000$) — the number of test cases. The description of the test cases follows.

Each test case consists of a single line containing two integers n and k ($2 \leq n \leq 10^6$, $1 \leq k \leq n$) — the parameters described above.

It is guaranteed that the sum of n over all test cases does not exceed 10^7.

Output

The first line of output for each test case should contain a single integer m ($1 \leq m \leq 25$) — the size of your chosen sequence.

The second line of output for each test case should contain m integers a_i ($0 \leq a_i \leq 10^9$) — the elements of your chosen sequence.

If there are multiple solutions, print any.

Standard Input	Standard Output
5 2 2 6 1 8 8 9 3 10 7	1 1 5 2 3 4 5 6 7 1 1 1 1 1 1 1 4 7 1 4 1 4 1 2 8 3

Note

In the first example, we just need a subsequence that adds up to 1, but not one that adds up to 2. So the array $a = [1]$ suffices.

In the second example, all elements are greater than $k = 1$, so no subsequence adds up to 1. Every other integer between 1 and n is present in the array, so there is a subsequence of size 1 adding up to each of those numbers.

,

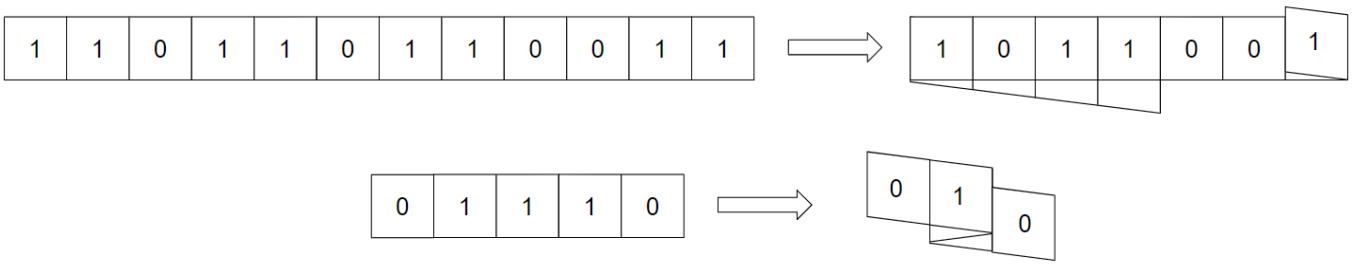
E. Folding Strip

Input file: standard input
Output file: standard output
Time limit: 2 seconds
Memory limit: 256 megabytes

You have a strip of paper with a binary string s of length n. You can fold the paper in between any pair of adjacent digits.

A set of folds is considered *valid* if after the folds, all characters that are on top of or below each other match. Note that all folds are made at the same time, so the characters don't have to match in between folds.

For example, these are valid foldings of $s = \texttt{110110110011}$ and $s = \texttt{01110}$:



The length of the folded strip is the length seen from above after all folds are made. So for the two above examples, after the folds shown above, the lengths would be 7 and 3, respectively.

Notice that for the above folding of $s = \texttt{01110}$, if we made either of the two folds on their own, that would not be a valid folding. However, because we don't check for validity until all folds are made, this folding is valid.

After performing a set of valid folds, what is the minimum length strip you can form?

Input

The first line of the input contains a single integer t ($1 \le t \le 10^4$) — the number of test cases. The description of the test cases follows.

The first line of each test case contains a single integer n ($1 \le n \le 2 \cdot 10^5$) — the size of the strip.

The second line of each test case contains a string s of n characters '$\texttt{0}$' and '$\texttt{1}$' — a description of the digits on the strip.

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 integer — the minimum possible length of the strip after a valid folding.

Standard Input	Standard Output
6 6 101101 1 0 12 110110110011 5	3 1 3 3 1 2

01110 4 1111 2 01	

Note

For the first example case, one optimal folding is to fold the strip in the middle, which produces a strip of length 3.

The third and fourth example cases correspond to the images above. Note that the folding shown above for $s = \texttt{110110110011}$ is not of minimal length.

,

F. Missing Subarray Sum

Input file: standard input
Output file: standard output
Time limit: 5 seconds
Memory limit: 256 megabytes

There is a hidden array a of n positive integers. You know that a is a **palindrome**, or in other words, for all $1 \leq i \leq n$, $a_i = a_{n+1-i}$. You are given the sums of all but one of its distinct subarrays, in arbitrary order. The subarray whose sum is not given can be any of the $\frac{n(n+1)}{2}$ distinct subarrays of a.

Recover any possible palindrome a. The input is chosen such that there is always at least one array a that satisfies the conditions.

An array b is a subarray of a if b can be obtained from a by the deletion of several (possibly, zero or all) elements from the beginning and several (possibly, zero or all) elements from the end.

Input

The first line of the input contains a single integer t ($1 \leq t \leq 200$) — the number of test cases. The description of the test cases follows.

The first line of each test case contains a single integer n ($3 \leq n \leq 1000$) — the size of the array a.

The next line of each test case contains $\frac{n(n+1)}{2} - 1$ integers s_i ($1 \leq s_i \leq 10^9$) — all but one of the subarray sums of a.

It is guaranteed that the sum of n over all test cases does not exceed 1000.

Additional constraint on the input: There is always at least one valid solution.

Hacks are disabled for this problem.

Output

For each test case, print one line containing n positive integers $a_1, a_2, \cdots a_n$ — any valid array a. Note that a must be a palindrome.

If there are multiple solutions, print any.

Standard Input	Standard Output
7 3 1 2 3 4 1 4 18 2 11 9 7 11 7 2 9 4 5 10 5 16 3 3 13 8 8 4 8 10 4 6 4 20 14 14 6 5 1 2 3 4 5 4 3 2 1 1 2 3 2 1 5 1 1 2 2 2 3 3 3 3 4 5 5 6 8	1 2 1 7 2 2 7 3 5 5 3 6 4 4 6 1 1 1 1 1 2 1 2 1 2 500000000 500000000 500000000

3 500000000 1000000000 500000000 500000000 1000000000	

Note

For the first example case, the subarrays of $a = [1, 2, 1]$ are:

- $[1]$ with sum 1,
- $[2]$ with sum 2,
- $[1]$ with sum 1,
- $[1, 2]$ with sum 3,
- $[2, 1]$ with sum 3,
- $[1, 2, 1]$ with sum 4.

So the full list of subarray sums is $1, 1, 2, 3, 3, 4$, and the sum that is missing from the input list is 3.
For the second example case, the missing subarray sum is 4, for the subarray $[2, 2]$.

For the third example case, the missing subarray sum is 13, because there are two subarrays with sum 13 ($[3, 5, 5]$ and $[5, 5, 3]$) but 13 only occurs once in the input.