Codeforces Global Round 26

A. Strange Splitting

1 second, 256 megabytes

Define the *range* of a non-empty array to be the maximum value minus the minimum value. For example, the range of $[1, 4, 2]$ is $4 - 1 = 3$.

You are given an array $a_1, a_2, \ldots, a_n$ of length $n \geq 3$. **It is guaranteed a is sorted.**

You have to color each element of a red or blue so that:

- the range of the red elements **does not equal** the range of the blue elements, and
- there is at least one element of each color.

If there does not exist any such coloring, you should report it. If there are multiple valid colorings, you can print any of them.

Input

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

The first line of each test case contains an integer n ($3 \leq n \leq 50$) — the length of the array.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($1 \leq a_i \leq 10^9$). It is guaranteed $a_1 \leq a_2 \leq \ldots \leq a_{n-1} \leq a_n$.

Output

For each test case, if it is impossible to color a to satisfy all the constraints, output `NO`.

Otherwise, first output `YES`.

Then, output a string s of length n. For $1 \leq i \leq n$, if you color a_i red, s_i should be `R`. For $1 \leq i \leq n$, if you color a_i blue, s_i should be `B`.

input

```
7
4
1 1 2 2
5
1 2 3 4 5
3
3 3 3
4
1 2 2 2
3
1 2 2
3
1 1 2
3
1 9 84
```

output

```
YES
RBRR
YES
BBRBB
NO
YES
RBBR
YES
RRB
YES
BRR
YES
BRB
```

In the first test case, given the array $[1, 1, 2, 2]$, we can color the second element blue and the remaining elements red; then the range of the red elements $[1, 2, 2]$ is $2 - 1 = 1$, and the range of the blue elements $[1]$ is $1 - 1 = 0$.

In the second test case, we can color the first, second, fourth and fifth elements $[1, 2, 4, 5]$ blue and the remaining elements $[3]$ red.

The range of the red elements is $3 - 3 = 0$ and the range of the blue elements is $5 - 1 = 4$, which are different.

In the third test case, it can be shown there is no way to color $a = [3, 3, 3]$ to satisfy the constraints.

B. Large Addition

1 second, 256 megabytes

A digit is *large* if it is between 5 and 9, inclusive. A positive integer is *large* if all of its digits are large.

You are given an integer x. Can it be the sum of two *large* positive integers with the **same number of digits**?

Input

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

The only line of each test case contains a single integer x ($10 \leq x \leq 10^{18}$).

Output

For each test case, output `YES` if x satisfies the condition, and `NO` otherwise.

You can output `YES` and `NO` in any case (for example, strings `yES`, `yes`, and `Yes` will be recognized as a positive response).

input

```
11
1337
200
1393938
1434
98765432123456789
11111111111111111
420
1984
10
69
119
```

output

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

In the first test case, we can have $658 + 679 = 1337$.

In the second test case, it can be shown that no numbers of equal length and only consisting of large digits can add to 200.

In the third test case, we can have $696\,969 + 696\,969 = 1\,393\,938$.

In the fourth test case, we can have $777 + 657 = 1434$.

C1. Magnitude (Easy Version)

2 seconds, 256 megabytes

The two versions of the problem are different. You may want to read both versions. You can make hacks only if both versions are solved.

You are given an array a of length n. Start with $c = 0$. Then, for each i from 1 to n (in increasing order) do **exactly one** of the following:

- Option 1: set c to $c + a_i$.
- Option 2: set c to $|c + a_i|$, where $|x|$ is the absolute value of x.

Let the maximum final value of c after the procedure described above be equal to k. Find k.

Input

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

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

The second line of each case contains n integers $a_1, a_2, a_3, \ldots, a_n$ ($-10^9 \le a_i \le 10^9$).

The sum of n over all test cases does not exceed $3 \cdot 10^5$.

Output

For each test case, output a single integer — the value of k.

input
5 4 10 -9 -3 4 8 1 4 3 4 1 4 3 4 3 -1 -2 -3 4 -1000000000 1000000000 1000000000 1000000000 4 1 9 8 4

output
6 24 6 4000000000 22

In the first test case, if we set c to its absolute value every time we add to it, we end up with 6. It can be shown that this is the maximum result.

In the second test case, taking the absolute value will never change anything, so we can just sum the array without doing anything to get 24.

In the third test case, it is optimal to wait until the end to set c to its absolute value, resulting in an answer of 6.

C2. Magnitude (Hard Version)

2 seconds, 256 megabytes

The two versions of the problem are different. You may want to read both versions. You can make hacks only if both versions are solved.

You are given an array a of length n. Start with $c = 0$. Then, for each i from 1 to n (in increasing order) do **exactly one** of the following:

- Option 1: set c to $c + a_i$.
- Option 2: set c to $|c + a_i|$, where $|x|$ is the absolute value of x.

Let the maximum final value of c after the procedure described above be equal to k. Find the number of unique procedures that result in $c = k$. Two procedures are different if at any index i, one procedure chose option 1 and another chose option 2, even if the value of c is equal for both procedures after that turn.

Since the answer may be large, output it modulo $998\,244\,353$.

Input

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

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

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($-10^9 \le a_i \le 10^9$).

The sum of n over all test cases does not exceed $3 \cdot 10^5$.

Output

For each test case, output a single integer — the number of unique procedures that result in $c = k$, modulo $998\,244\,353$.

input
5 4 2 -5 3 -3 8 1 4 3 4 1 4 3 4 3 -1 -2 -3 4 -1000000000 1000000000 1000000000 1000000000 4 1 9 8 4

output
12 256 1 8 16

In the first test case, it can be shown that our maximal final value of c is 3. There are 12 ways to achieve this because in order to get 3, we have to take absolute value at indices 2 or 4, or both, resulting in 3 ways. For the other two indices, it doesn't change the value whether we take absolute value or not, so we have $2 \cdot 2 = 4$ ways for them. In total, we have $3 \cdot 4 = 12$ ways.

In the second test case, taking the absolute value will never change anything, so we can either take absolute value or not, for every index. This gives us $2^8 = 256$ possible ways.

D. "a" String Problem

2 seconds, 256 megabytes

You are given a string s consisting of lowercase Latin characters. Count the number of nonempty strings $t \neq$ "a" such that it is possible to partition [†] s into some substrings satisfying the following conditions:

- each substring either equals t or "a", and
- at least one substring equals t.

[†] A *partition* of a string s is an ordered sequence of some k strings $t_1, t_2, \ldots, t_k$ (called *substrings*) such that $t_1 + t_2 + \ldots + t_k = s$, where $+$ represents the concatenation operation.

Input

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

The only line of each test case contains a string s consisting of lowercase Latin characters ($2 \le |s| \le 2 \cdot 10^5$).

The sum of $|s|$ over all test cases does not exceed $3 \cdot 10^5$.

Output

For each test case, output a single integer — the number of nonempty strings $t \neq$ "a" that satisfy all constraints.

input
8 aaaaa baba cabacb aaabaaa bitset ab abbaaaabbb yearnineteeneightyfour

output

```
4
4
1
16
1
2
3
1
```

In the first test case, t can be "aa", "aaa", "aaaa", or the full string.

In the second test case, t can be "b", "bab", "ba", or the full string.

In the third test case, the only such t is the full string.

E. Shuffle

2 seconds, 256 megabytes

Two hungry red pandas, Oscar and Lura, have a tree T with n nodes. They are willing to perform the following **shuffle procedure** on the whole tree T **exactly once**. With this shuffle procedure, they will create a new tree out of the nodes of the old tree.

1. Choose any node V from the original tree T. Create a new tree T_2, with V as the root.
2. Remove V from T, such that the original tree is split into one or more subtrees (or zero subtrees, if V is the only node in T).
3. Shuffle each subtree with the same procedure (again choosing any node as the root), then connect all shuffled subtrees' roots back to V to finish constructing T_2.

After this, Oscar and Lura are left with a new tree T_2. They can only eat leaves and are very hungry, so please find the maximum number of leaves over all trees that can be created in **exactly one** shuffle.

Note that leaves are all nodes with degree 1. Thus, the root may be considered as a leaf if it has only one child.

Input

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

The first line of every test case contains a single integer n ($2 \le n \le 2 \cdot 10^5$) — the number of nodes within the original tree T.

The next $n - 1$ lines each contain two integers u and v ($1 \le u, v \le n$) — an edge within the original tree T. The given edges form a tree.

The sum of n over all test cases does not exceed $3 \cdot 10^5$.

Output

For each test case, output a single integer — the maximum number of leaves achievable with exactly one shuffle procedure on the whole tree.

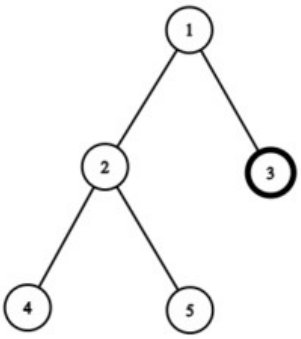
input

```
4
5
1 2
1 3
2 4
2 5
5
1 2
2 3
3 4
4 5
6
1 2
1 3
1 4
1 5
1 6
10
9 3
8 1
10 6
8 5
7 8
4 6
1 3
10 1
2 7
```

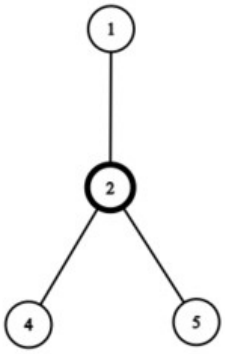
output

```
4
3
5
6
```

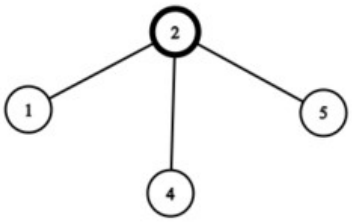
In the first test case, it can be shown that the maximum number of leaves is 4. To accomplish this, we can start our shuffle with selecting node 3 as the new root.



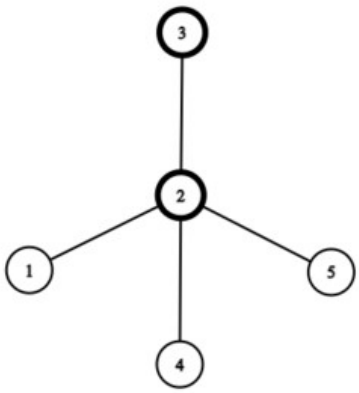
Next, we are left only with one subtree, in which we can select node 2 to be the new root of that subtree.



This will force all 3 remaining nodes to be leaves, and once we connect them back to our new root, the shuffled subtree looks like this:



We connect the shuffled subtree back to our new root of our new tree. Our final tree has four leaves (including the root), and looks like this:



In our second test case, we have a line of five nodes. It can be shown that the maximum number of leaves after one shuffle is 3. We can start off with node 2, which forces node 1 to become a leaf. Then, if we select node 4 on the right side, we will also have nodes 3 and 5 as leaves.

The third test case is a star graph with six nodes. The number of leaves cannot increase, thus our answer will be 5 (if we start the shuffling with the original root node).

F. Reconstruction

2 seconds, 256 megabytes

There is a hidden array $a_1, a_2, \ldots, a_n$ of length n whose elements are integers between $-m$ and m, inclusive.

You are given an array $b_1, b_2, \ldots, b_n$ of length n and a string s of length n consisting of the characters P, S, and ?.

For each i from 1 to n inclusive, we must have:

- If $s_i = \texttt{P}$, b_i is the sum of a_1 through a_i.
- If $s_i = \texttt{S}$, b_i is the sum of a_i through a_n.

Output the number of ways to replace all ? in s with either P or S such that there exists an array $a_1, a_2, \ldots, a_n$ with elements not exceeding m by absolute value satisfying the constraints given by the array $b_1, b_2, \ldots, b_n$ and the string s.

Since the answer may be large, output it modulo $998\,244\,353$.

Input

The first line contains a single integer t ($1 \le t \le 10^3$) — the number of test cases.

The first line of each test case contains two integers n and m ($2 \le n \le 2 \cdot 10^3$, $2 \le m \le 10^9$) — the length of the hidden array $a_1, a_2, \ldots, a_n$ and the maximum absolute value of an element a_i.

The second line of each test case contains a string s of length n consisting of characters P, S, and ?.

The third line of each test case contains n integers $b_1, b_2, \ldots, b_n$ ($|b_i| \le m \cdot n$).

The sum of n over all test cases does not exceed $5 \cdot 10^3$.

Output

For each test case, output a single integer — the number of ways to replace all ? in s with either P or S that result in the existence of a valid array $a_1, a_2, \ldots, a_n$, modulo $998\,244\,353$.

input

```
6
4 10
PSPP
1 9 8 10
4 1000000000
????
1 1 1 4000000000
8 1000000000
?P?SSP?P
-857095623 -1424391899 -851974476 673437144 471253851 -543483033
364945701 -178537332
4 7
PPSS
4 2 1 3
9 20
?????????
1 2 3 4 5 6 7 8 9
3 1000000000
P??
-145463248 -974068460 -1287458396
```

output

```
1
0
2
1
14
1
```

In the first test case, we can see that the following array satisfies all constraints, thus the answer is 1:

1. P — $[\mathbf{1}, 3, 4, 2]$: sum of 1.
2. S — $[1, \mathbf{3}, 4, 2]$: sum of 9.
3. P — $[1, 3, \mathbf{4}, 2]$: sum of 8.
4. P — $[1, 3, 4, \mathbf{2}]$: sum of 10.

In the second test case, it can be shown that no array a with all $|a_i| \le m = 10^9$ satisfies all constraints.

G. Magic Trick II

2 seconds, 256 megabytes

The secret behind Oscar's first magic trick has been revealed! Because he still wants to impress Lura, he comes up with a new idea: he still wants to sort a permutation $p_1, p_2, \ldots, p_n$ of $[1, 2, \ldots, n]$.

This time, he chooses an integer k. He wants to sort the permutation in non-decreasing order using the following operation several times:

1. Pick a continuous subarray of length k and remove it from p.
2. Insert the continuous subarray back into p at any position (perhaps, in the very front or the very back).

To be as impressive as possible, Oscar would like to choose the maximal value of k such that he can sort his permutation. Please help him find the maximal k as well as a sequence of operations that will sort the permutation. You don't need to minimize the number of operations, but you are allowed to use at most $5n^2$ operations.

We have a proof that, for the maximal k such that you can sort the permutation in any number of operations, you can also sort it in at most $5n^2$ operations.

Input

The first line contains a single integer t ($1 \le t \le 10^3$) — the number of test cases.

The first line of each test case contains a single integer n ($5 \le n \le 10^3$) — the length of the permutation.

The second line of each test case contains a permutation $p_1, p_2, \ldots, p_n$ of $[1, 2, \ldots, n]$.

The sum of n over all test cases does not exceed $2 \cdot 10^3$.

Output

For each test case, first output the chosen value of k on a new line ($1 \le k \le n$).

Then, output a single integer m — the number of operations used ($0 \le m \le 5n^2$).

Then, on each of the next m lines, output the operations denoted by two integers i and j ($1 \le i, j \le n - k + 1$), representing an operation where you remove the subarray starting from index i and replace it back into p at index j.

input
3 5 5 1 2 3 4 5 2 3 5 4 1 6 1 2 3 4 5 6

output
4 1 2 1 3 2 1 3 2 1 6 0

In the first test case, it is enough to move the last four numbers to the front.

In the second test case, it can be shown that we cannot have $k = 4$ or $k = 5$. With $k = 3$, we can move the first three numbers to the end, and then the middle three to the front to sort the permutation.

In the third test case, the permutation is already sorted. We can have $k = 6$ and use no operations.

H. Tower Capturing

2 seconds, 256 megabytes

There are n towers at n distinct points $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, such that no three are collinear and no four are concyclic. Initially, you own towers (x_1, y_1) and (x_2, y_2), and you want to capture all of them. To do this, you can do the following operation any number of times:

- Pick two towers P and Q you own and one tower R you don't own, such that the circle through P, Q, and R contains **all** n towers inside of it.
- Afterwards, capture all towers in or on triangle $\triangle PQR$, including R itself.

An *attack plan* is a series of choices of R ($R_1, R_2, \ldots, R_k$) using the above operations after which you capture all towers. Note that two attack plans are considered different only if they differ in their choice of R in some operation; in particular, two attack plans using the same choices of R but different choices of P and Q are considered the same.

Count the number of attack plans of **minimal length**. Note that it might not be possible to capture all towers, in which case the answer is 0.

Since the answer may be large, output it modulo $998\,244\,353$.

Input

The first line contains a single integer t ($1 \le t \le 250$) — the number of test cases.

The first line of each test case contains a single integer n ($4 \le n \le 100$) — the number of towers.

The i-th of the next n lines contains two integers x_i and y_i ($-10^4 \le x_i, y_i \le 10^4$) — the location of the i-th tower. Initially, you own towers (x_1, y_1) and (x_2, y_2).

All towers are at distinct locations, no three towers are collinear, and no four towers are concyclic.

The sum of n over all test cases does not exceed 1000.

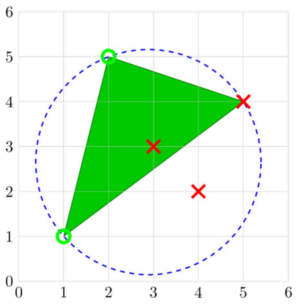
Output

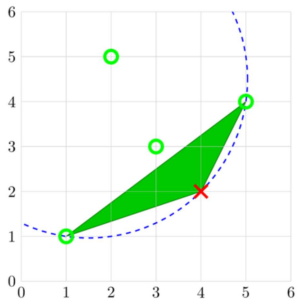
For each test case, output a single integer — the number of attack plans of minimal length after which you capture all towers, modulo $998\,244\,353$.

input
3 5 1 1 2 5 3 3 4 2 5 4 6 1 1 3 3 1 2 2 1 3 10000 19 84 7 2 7 -4 -3 -3 6 3 1 -5 2 1 -4 -1 7

output
1 0 10

In the first test case, there is only one possible attack plan of shortest length, shown below.





- Use the operation with $P =$ tower 1, $Q =$ tower 2, and $R =$ tower 5. The circle through these three towers contains all towers inside of it, and as a result towers 3 and 5 are captured.
- Use the operation with $P =$ tower 5, $Q =$ tower 1, and $R =$ tower 4. The circle through these three towers contains all towers inside of it, and as a result tower 4 is captured.

In the second case, for example, you can never capture the tower at $(3, 10\,000)$.