Problem A. Adjacent Product Sum

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

You have n numbers $a_1, a_2, \ldots, a_n$. You want to arrange them in a circle so as to maximize the sum of products of pairs of adjacent numbers.

Formally, you want to find a permutation $b_1, b_2, \ldots, b_n$ of $a_1, a_2, \ldots, a_n$ such that $b_1b_2 + b_2b_3 + \ldots + b_{n-1}b_n + b_nb_1$ is maximal.

Find this maximum value.

Input

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

The first line of each test case contains a single integer n ($3 \le n \le 2 \cdot 10^5$) — the number of numbers.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($-10^6 \le a_i \le 10^6$).

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 possible value of the expression $b_1b_2 + b_2b_3 + \ldots + b_{n-1}b_n + b_nb_1$ over all permutations $b_1, b_2, \ldots, b_n$ of $a_1, a_2, \ldots, a_n$.

Example

standard input	standard output
4 3 1 2 3 6 1 1 1 1 0 0 5 100000 100000 100000 100000 -100000 5 1 2 3 4 5	11 3 10000000000 48

Note

In the first test case, there is only one way to arrange the numbers in a circle (not counting rotations and symmetries) — $(1, 2, 3)$. The sum of products of pairs of adjacent numbers is $1 \cdot 2 + 2 \cdot 3 + 3 \cdot 1 = 11$.

In the second test case, one of the optimal arrangements is $(1, 1, 1, 1, 0, 0)$. For it this sum is equal to $1 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 + 1 \cdot 0 + 0 \cdot 0 + 0 \cdot 1 = 3$.

In the third test case, there is a unique (up to rotations and symmetries) way to arrange the numbers in a circle: $(100000, 100000, 100000, 100000, -100000)$, the answer is $100000^2 = 10^{10}$. Note that the answer may not fit into int32.

In the fourth test case, one of the optimal permutations is $(1, 2, 4, 5, 3)$, the answer for which is $1 \cdot 2 + 2 \cdot 4 + 4 \cdot 5 + 5 \cdot 3 + 3 \cdot 1 = 2 + 8 + 20 + 15 + 3 = 48$.

Problem B. Binary Arrays and Sliding Sums

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

You are given two integers n, k ($2 \le k < n$).

For an array $a_1, a_2, \ldots, a_n$, which **consists only of zeros and ones**, we define the array $f(a)$ of length n as follows:

- $f(a)_i = a_i + a_{i+1} + \ldots + a_{i+k-2} + a_{i+k-1}$ (here we assume that $a_{n+i} = a_i$, i.e. numbers are arranged in a circle).

  For example, for $n = 4, k = 2$, $f([0, 1, 1, 0]) = [1, 2, 1, 0]$.

Consider all 2^n possible arrays a, and for each of them, find $f(a)$. How many different arrays are there among them? Since the answer may be very large, print the number of these arrays modulo 998244353.

Two arrays are considered different if they differ in at least one position.

Input

The first line contains a single integer t ($1 \le t \le 10^5$) — the number of test cases. The description of test cases follows.

The first line of each test case contains two integers n, k ($2 \le k < n \le 10^6$).

Output

For each test case print the number of different arrays among $f(a)$, modulo 998244353.

Example

standard input	standard output
4 3 2 4 2 42 3 123123 123	8 15 780086989 126500246

Note

For $n = 3, k = 2$, there are 8 different arrays a of ones and zeros. The corresponding arrays $f(a)$ are pairwise distinct for them.

For $n = 4, k = 2$, there are 16 distinct arrays a of ones and zeros. The only pair of matching arrays $f(a)$ is $[0, 1, 0, 1]$ and $[1, 0, 1, 0]$: for both arrays $f(a)$ is $[1, 1, 1, 1]$.

Problem C. Count Hamiltonian Cycles

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

You are given a string s of length $2n$, containing n characters `W` and n characters `B`.

Let's build a graph on $2n$ nodes. If $s_i \neq s_j$ for some $1 \leq i < j \leq 2n$, then there is an edge of weight $|i - j|$ between nodes i and j in this graph. There are no other edges.

Find the number of shortest Hamiltonian cycles in this graph. As this number can be very large, output it modulo 998244353.

As a reminder, a Hamiltonian cycle is a cycle that visits each node exactly once. The length of the cycle is equal to the sum of the weights of its edges. Two cycles are called different if there is an edge that one contains and the other doesn't.

Input

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

The first line of each test case contains a single integer n ($2 \leq n \leq 10^6$).

The second line of each test case contains a string s of length $2n$, containing n characters `W` and n characters `B`.

It is guaranteed that the sum of n over all test cases does not exceed 10^6.

Output

For each test case, output the number of shortest Hamiltonian cycles in this graph, modulo 998244353.

Example

standard input	standard output
3 2 WWBB 3 WBWBWB 7 WWWWBWBBWWBBBB	1 2 62208

Note

In the first test case, the graph has 4 edges: $(1, 3)$ with weight 2, $(1, 4)$ with weight 3, $(2, 3)$ with weight 1, and $(2, 4)$ with weight 2.

There is a unique Hamiltonian cycle here: $1 \to 3 \to 2 \to 4 \to 1$ (Note that, for example, cycle $1 \to 4 \to 2 \to 3 \to 1$ contains the same set of edges, so we have already counted it).

Problem D. Distance Parities

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

Andrii had a connected graph with n vertices. For every two different vertices i and j of this graph, he calculated the length of the shortest path between them — $d_{i,j}$. Unfortunately, then Andrii lost the graph and forgot the numbers $d_{i,j}$. But he remembered the parity of all numbers $d_{i,j}$.

So for every two different vertices i, j Andrii told you $a_{i,j} = d_{i,j} \bmod 2$. Construct an example of a graph that Andrii could have had, or determine that such a graph does not exist and Andrii is lying to you.

Input

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

The first line of each test case contains one integer n ($2 \le n \le 500$) — the number of vertices.

The i-th of the next n lines contains a binary string s_i of length n. The j-th character of this string is `0` if $a_{i,j} = 0$, and `1` if $a_{i,j} = 1$.

It is guaranteed that $a_{i,i} = 0$ for all $1 \le i \le n$, and $a_{i,j} = a_{j,i}$ for all $1 \le i < j \le n$.

It is guaranteed that the sum of n^2 over all test cases does not exceed 250000.

Output

For each test case, if such a graph does not exist, print `NO`.

Otherwise, print `YES`. On the next line print a single integer m ($n - 1 \le m \le \frac{n(n-1)}{2}$) — the number of edges. In the i-th of the next m lines print two numbers u_i, v_i ($1 \le u_i, v_i \le n, u_i \ne v_i$), denoting the edge between the vertices u_i and v_i.

All edges must be pairwise distinct. The graph must be connected.

You can print `YES` and `NO` in any case (e.g. the strings `yEs`, `yes`, `Yes` will be taken as a positive answer).

Example

standard input	standard output
3 3 011 101 110 4 0100 1000 0001 0010 5 01010 10101 01010 10101 01010	YES 3 1 2 1 3 2 3 NO YES 4 1 2 2 3 3 4 4 5

Note

In the first test case, such a graph on three vertices exists — you can just take a triangle. All pairwise distances are equal to 1 and hence odd.

It can be shown that in the second test case, such a graph does not exist.

In the third test case, we have a chain with edges $(1, 2), (2, 3), (3, 4), (4, 5)$. In it, the distance between vertices i, j is odd if and only if i and j have different parity.

Problem E. Excellent XOR Problem

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

For an array $[b_1, b_2, \ldots, b_k]$ of integers, let's define its **weight** as the $\oplus$ of all its elements.

Here $\oplus$ denotes the **bitwise exclusive OR** operation. For example, $13 \oplus 6 = 11$, because in binary, $13 = \texttt{1101}$ and $6 = \texttt{0110}$, so their $\oplus$ is $\texttt{1011} = 11$. The weight of array $[13, 1, 4]$, for example, is 8.

You are given an array $[a_1, a_2, \ldots, a_n]$ of integers. We want to divide it into several (**more than one**) consecutive subarrays whose weights are distinct. Determine if this is possible. If it is possible, find one of such partitions.

Input

The first line 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 ($2 \le n \le 2 \cdot 10^5$) — the length of the array.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($0 \le a_i < 2^{30}$) — 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, if no such partitioning exists, print `NO`.

Otherwise, print `YES`. On the following line, print a single integer k ($2 \le k \le n$) — the number of subarrays into which you are splitting a.

On the i-th of the next k lines print two numbers l_i, r_i ($1 \le l_i \le r_i \le n$), denoting that the i-th of your arrays is $[a_{l_i}, a_{l_i+1}, \ldots, a_{r_i}]$. You can print these subarrays in any order, but each number from 1 to n must appear in **exactly one** of the segments $[l_i, r_i]$.

You can print `YES` and `NO` in any case (e.g. the strings `yEs`, `yes`, `Yes` will be taken as a positive answer).

Example

standard input	standard output
4 2 0 0 3 1 2 3 5 16 8 4 2 1 6 42 42 42 42 42 42	NO YES 3 1 1 2 2 3 3 YES 2 1 1 2 5 NO

Note

In the first test case, there is no way to split $[0, 0]$ into at least two subarrays with distinct $\oplus$s.

In the second test case, you can split array $[1, 2, 3]$ into 3 subarrays $[1], [2], [3]$ correspondingly, with $\oplus$s $1, 2, 3$ correspondingly.

Problem F. F*** 3-Colorable Graphs

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

A graph is called k-colorable if it's possible to color each its node in one of k colors so that for any two nodes u and v, which are connected by an edge, their colors will be different.

You are given a **connected bipartite** graph, where one part has n nodes, numbered from 1 to n, and the second part has n nodes, numbered from $n+1$ to $2n$. There is no edge between any two nodes from the first part, and there is no edge between any two nodes from the second part.

(A graph is called bipartite if its nodes can be split into two nonempty parts, such that each edge connects nodes from different parts).

This graph, of course, is 2-colorable: you can color all nodes from the first part in color 1 and from the second in color 2. But you don't like that. You don't want this graph to be 2-colorable. You don't even want it to be 3-colorable!

You want to add some edges to this graph so that it stops being 3-colorable (in particular, you can add edges between two nodes from the same part). What's the smallest number of edges you have to add?

Input

The first line contains two integers n, m ($2 \le n \le 10^4, 2n-1 \le m \le min(n^2, 2 \cdot 10^5)$) — the size of each part and the number of edges correspondingly.

The i-th of the next m lines contains two integers u_i, v_i ($1 \le u_i \le n$, $n+1 \le v_i \le 2n$), denoting the edge (u_i, v_i).

It's guaranteed that no edge will appear more than once. It's guaranteed that the graph on these edges is connected.

Output

Output a single integer — the smallest number of edges that you have to add to this graph so that it becomes not 3-colorable.

Examples

standard input	standard output
2 4 1 3 1 4 2 3 2 4	2
3 5 1 4 2 4 2 5 3 5 3 6	3

Note

In the first sample, you can add edges $(1, 2)$ and $(3, 4)$ to the graph. You will get a complete graph on 4 nodes, which is not 3-colorable.

In the second sample, you can't add less than 3 edges to make graph not 3-colorable.

Problem G. Graph Problem With Small n

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

You are given an undirected graph with n vertices. For each pair of vertices (i, j) $(i \neq j)$, determine whether there exists a Hamiltonian path starting at i and ending at j.

Recall that a Hamiltonian path is a path consisting of $n-1$ edges that passes through all vertices exactly once.

Input

The first line contains one integer n $(2 \leq n \leq 24)$ — the number of vertices in the graph.

The i-th of the next n lines contains a binary string s_i of length n. Its i-th character is always equal to `0`, and for $j \neq i$ its j-th character is equal to `1` if there is an edge between vertices i and j, and `0` otherwise.

It is guaranteed that for any $i \neq j$, the i-th character of the j-th line coincides with the j-th character of the i-th line.

Output

Print n lines. In i-th of them, print a binary string of length n. Its i-th character must be equal to `0`, and j-th character at $j \neq i$ must be equal to `1` if there is a Hamiltonian path between vertices i and j, and `0` otherwise.

Examples

standard input	standard output
4 0110 1010 1101 0010	0001 0001 0000 1100
6 010001 101000 010100 001010 000101 100010	010001 101000 010100 001010 000101 100010
4 0111 1011 1101 1110	0111 1011 1101 1110

Note

In the first example, the Hamiltonian path exists between pairs $(1, 4)$ and $(2, 4)$.

In the second example, the graph is a cycle of length 6. The Hamiltonian path here exists only between pairs of adjacent vertices.

In the third example, we have a complete graph with 4 vertices. There exists a Hamiltonian path between each pair of vertices.

Problem H. Help Me to Get This Published

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

A **Gallai coloring** of a complete graph on n nodes is a coloring of its edges, in which the following condition holds:

- There is no triangle whose edges are colored in 3 distinct colors.

The color degree $d(v)$ of node v is defined as the number of different colors that appear on the edges incident to v. Let's call sequence $(a_1, a_2, \ldots, a_n)$ a **valid degree sequence** if there exists some Gallai coloring of a complete graph on n nodes, in which $d(i) = a_i$ for all $1 \le i \le n$.

You are given some values of a_i, and some are equal to -1. Find the number of ways to replace elements of a, equal to -1, to obtain a valid degree sequence. As this number may be large, output it modulo 998244353.

Input

The first line of the input contains a single integer n ($2 \le n \le 100$).

The second line of the input contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le n - 1$, or $a_i = -1$). If $a_i \ne -1$, its value is given.

Output

Output a single integer — the number of ways to replace elements of a, equal to -1, to obtain a valid degree sequence, modulo 998244353.

Examples

standard input	standard output
2 1 -1	1
3 -1 -1 -1	4
6 5 -1 -1 -1 -1 -1	120

Note

In the first sample, the only valid degree sequence is $(1, 1)$.

In the second sample, the only valid degree sequences are $(1, 1, 1), (1, 2, 2), (2, 1, 2), (2, 2, 1)$, where the first one corresponds to the case, when all edges have the same color, and the next three correspond to the cases, when some two edges have the same color.

Problem I. Increasing Grid

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

There is a table $n \times m$, which we want to fill with integers. We denote the number in the cell at the intersection of i-th row and j-th column as $a_{i,j}$.

We want the following conditions to hold:

- $1 \le a_{i,j} \le n + m$ for all $1 \le i \le n$, $1 \le j \le m$.
- $a_{i-1,j} < a_{i,j}$ for all $1 \le i \le n - 1$, $1 \le j \le m$.
- $a_{i,j-1} < a_{i,j}$ for all $1 \le i \le n$, $1 \le j \le m - 1$.

In other words, we want to fill the table with numbers from 1 to $n + m$, so that the numbers in each row and column are increasing.

Also, we know the values of numbers in some cells of the table. Find the number of ways to choose the values of the numbers that we do not know, so that all the conditions above are satisfied. Since this number can be very large, print it modulo 998244353.

Input

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

The first line of each test case contains two integers n, m ($1 \le n, m \le 2 \cdot 10^5, 1 \le n \cdot m \le 2 \cdot 10^5$) — the size of the table. Then there are n rows.

The i-th of the next n lines contains m integers $a_{i,1}, a_{i,2}, \ldots, a_{i,m}$ ($1 \le a_{i,j} \le n + m$ or $a_{i,j} = -1$). If $a_{i,j} \ne -1$, then the number at the intersection of the i-th row and j-th column is known (and equal to $a_{i,j}$); otherwise, it must be found.

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

Output

For each test case, print a single integer — the number of ways to fill in the unknown values of the table so that all conditions are satisfied.

Example

standard input	standard output
4 2 3 1 2 -1 -1 4 5 4 4 -1 -1 -1 -1 -1 -1 -1 -1 -1 4 4 -1 -1 -1 -1 -1 4 4 -1 -1 -1 -1 -1 -1 -1 -1 -1 4 5 -1 -1 -1 -1 -1 3 5 1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1	4 0 17 55

Note

In the first test case, we only need to choose the values of $a_{2,1}$ and $a_{1,3}$. $a_{2,1}$ can take the values $2, 3$, and $a_{1,3}$ — $3, 4$. There are 4 options in total.

In the second test case, there is no such table because the condition $a_{3,2} < a_{3,3}$ is already violated.

Problem J. Jewel of Data Structure Problems

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

The array $a_1, a_2, \ldots, a_m$ of integers is called **odd** if it has an odd number of inversions, and **even** otherwise. Recall that an inversion is a pair (i, j) with $1 \le i < j \le m$ such that $a_i > a_j$. For example, in the array $[2, 4, 1, 3]$ there are 3 inversions: $(1, 3), (2, 3), (2, 4)$ (since $a_1 > a_3, a_2 > a_3, a_2 > a_4$), so it is **odd**.

Given a permutation $p_1, p_2, \ldots, p_n$ of integers from 1 to n, we call its beauty the length of its longest **odd** subsequence, if it exists, otherwise -1. For example, the beauty of the permutation $(1, 2, 3)$ is -1, because each of its subsequences is **even**, the beauty of $(4, 1, 2, 3)$ is 4, because the whole permutation is **odd**, and the beauty of $(4, 1, 3, 2)$ is 3, because the whole permutation is **even**, and the subsequence $(4, 3, 2)$ is **odd**.

We are given an initial permutation $p_1, p_2, \ldots, p_n$. There will be q update requests to it. After the i-th request we will have to swap p_{u_i} and p_{v_i}.

Find the beauty of the permutation after each request.

Recall that an array b is a subsequence of c if b can be obtained from c by removing some (possibly none or all) elements.

Recall that a permutation of the numbers from 1 to n is an array of length n containing each number from 1 to n exactly once.

Input

The first line contains two integers n, q ($1 \le n, q \le 2 \cdot 10^5$) — the permutation length and the number of queries, respectively.

The next line contains n integers $p_1, p_2, \ldots, p_n$ ($1 \le p_i \le n$, all p_i are pairwise distinct) — the initial permutation p.

The i-th of the following q lines contains two integers u_i, v_i ($1 \le u_i, v_i \le n$, $u_i \ne v_i$), indicating that after the i-th query you have to swap p_{u_i} and p_{v_i}.

Output

Print q integers — the permutation beauty after each update request.

Example

standard input	standard output
5 6 2 1 3 4 5 1 2 1 2 1 4 2 1 3 5 1 3	-1 5 4 5 3 5

Note

- After the first query, the permutation is $(1, 2, 3, 4, 5)$. There is no **odd** subsequence in it.
- After the second query, the permutation is $(2, 1, 3, 4, 5)$. The whole permutation is **odd**, because it has exactly one inversion.

- After the third query, the permutation is $(4, 1, 3, 2, 5)$. The whole permutation is **even**, but its subsequence $(4, 3, 2, 5)$ is **odd**.
- After the fourth query, the permutation is $(1, 4, 3, 2, 5)$. The entire permutation is **odd**.
- After the fifth query, the permutation is $(1, 4, 5, 2, 3)$. The entire permutation is **even**, and all its subsequences of length 4 are **even**, but the subsequence $(1, 5, 2)$ is **odd**.
- After the sixth query, the permutation is $(5, 4, 1, 2, 3)$. The entire permutation is **odd**.

Problem K. King of Swapping

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

You have a device for working with permutations $p_1, p_2, \ldots, p_n$ of integers from 1 to n. It can perform m operations. i-th operation is described by two integers a_i, b_i ($1 \le a_i, b_i \le n, a_i \ne b_i$). If you apply it, it will do the following: if $p_{a_i} > p_{b_i}$, then the device will swap p_{a_i}, p_{b_i}. Otherwise, it will do nothing.

You can apply these operations any number of times, in any order (you can use one operation more than once).

You are interested in whether you can use this device to get every permutation from every other permutation. In other words, determine whether for every two permutations $p_1, p_2, \ldots, p_n$ and $q_1, q_2, \ldots, q_n$ of the integers from 1 to n, there exists a sequence of operations that can be applied to p to obtain q.

Recall that the permutation of integers from 1 to n is an array of length n containing each integer from 1 to n exactly once.

Input

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

The first line of each test case contains two integers n, m ($1 \le n \le 2 \cdot 10^5, 0 \le m \le 2 \cdot 10^5$) —the length of permutations your device can handle and the number of operations it can perform.

The i-th of the m following lines contains two integers a_i, b_i ($1 \le a_i, b_i \le n, a_i \ne b_i$) describing the i-th operation: if $p_{a_i} > p_{b_i}$, the device can swap the elements p_{a_i}, p_{b_i} of the permutation.

It is guaranteed that all pairs (a_i, b_i) are pairwise distinct, but note that pairs (x, y) and (y, x) may occur simultaneously for some x, y.

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

Output

For each test case, print `YES` if you can obtain any permutation from any other permutation with this device, and `NO` otherwise.

You can print `YES` and `NO` in any case (e.g. the strings `yEs`, `yes`, `Yes` will be taken as a positive answer).

Example

standard input	standard output
5 2 1 1 2 3 4 1 2 2 1 1 3 3 1 5 4 1 2 2 3 3 4 4 5 5 6 3 5 5 1 1 2 4 3 2 4 4 1 4 6 3 1 2 3 2 1 3 2 1 2 1 3	NO YES NO YES NO

Note

In the first example, we can swap p_1 and p_2 if $p_1 > p_2$. Thus, we can get the permutation $(2, 1)$ from the permutation $(1, 2)$, but we cannot get the permutation $(1, 2)$ from the permutation $(2, 1)$, so the answer is `NO`.

In the second example, we can swap p_1, p_2 regardless of which one is larger (because we have both swap operations: with $a_i = 1, b_i = 2$ and with $a_i = 2, b_i = 1$). Also, we can swap p_1, p_3 regardless of which of them is greater. It is easy to show that in this case we can get from any permutation any other permutation, so the answer is `YES`.

Problem L. Least Annoying Constructive Problem

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

Consider a complete graph on n nodes. You have to arrange all its $\frac{n(n-1)}{2}$ edges on the circle in such a way that every $n-1$ consecutive edges on this circle form a tree.

It can be proved that such an arrangement is possible for every n. If there are many such arrangements, you can find any of them.

As a reminder, a tree on n nodes is a connected graph with $n-1$ edges.

Input

The only line of the input contains a single integer n ($3 \le n \le 500$).

Output

Output $\frac{n(n-1)}{2}$ lines. The i-th line should contain two integers u_i, v_i ($1 \le u_i < v_i \le n$). All pairs (u_i, v_i) have to be distinct, and for every i from 1 to $\frac{n(n-1)}{2}$, edges $(u_i, v_i), (u_{i+1}, v_{i+1}), \ldots, (u_{i+n-2}, v_{i+n-2})$ have to form a tree.

Here $u_{\frac{n(n-1)}{2}+i} = u_i, v_{\frac{n(n-1)}{2}+i} = v_i$ for every i.

Examples

standard input	standard output
3	1 2 2 3 1 3
4	1 2 3 4 2 3 1 4 1 3 2 4

Problem M. Most Annoying Constructive Problem

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

The array $a_1, a_2, \ldots, a_m$ of integers is called **odd** if it has an odd number of inversions, and **even** otherwise. Recall that an inversion is a pair (i, j) with $1 \le i < j \le m$ such that $a_i > a_j$. For example, in the array $[2, 4, 1, 3]$, there are 3 inversions: $(1, 3), (2, 3), (2, 4)$ (since $a_1 > a_3, a_2 > a_3, a_2 > a_4$), so it is **odd**.

Given n, k, determine if there exists a permutation of integers from 1 to n, which has exactly k odd subarrays.

An array b is a subarray of an array c if b can be obtained from c 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 contains a single integer t ($1 \le t \le 10^4$) — the number of test cases. The description of the test cases follows.

The only line of each test case contains two integers n, k ($1 \le n \le 1000, 0 \le k \le \frac{n(n-1)}{2}$).

It's guaranteed that the sum of n^2 over all test cases doesn't exceed $4 \cdot 10^6$.

Output

For every test case, if there is no such permutation, output `NO`.

Otherwise, output `YES`. In the next line, output n integers $p_1, p_2, \ldots, p_n$ ($1 \le p_i \le n$, all p_i are distinct) — the elements of your permutation.

Example

standard input	standard output
4 1 0 3 3 4 1 6 15	YES 1 YES 3 2 1 YES 1 3 4 2 NO

Note

In the first test case, the permutation is (1); all its subarrays are even.

In the second test case, the permutation is $(3, 2, 1)$. It has 3 odd subarrays: $[3, 2], [2, 1]$ with 1 inversion each, and $[3, 2, 1]$ with 3 inversions.

In the third test case, the permutation is $(1, 3, 4, 2)$. It has exactly 1 odd subarrays: $[4, 2]$ with 1 inversion.

It can be shown that no such permutation exists for the fourth test case.

Problem N. No Zero-Sum Subsegment

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

You are given integers A, B, C, D. Count the number of arrays of length $A + B + C + D$, such that:

- They contain exactly A elements equal to -2, exactly B elements equal to -1, exactly C elements equal to 1, exactly D elements equal to 2
- They contain no subarray with sum equal to 0.

As this number can be very large, output it modulo 998244353.

An array b is a subarray of an array c if b can be obtained from c 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 ($1 \le t \le 10^5$) — the number of test cases. The description of test cases follows.

The only line of each test case contains 4 integers A, B, C, D ($0 \le A, B, C, D \le 10^6$, $A + B + C + D > 0$).

Output

Output a single integer — answer to the problem.

Example

standard input	standard output
5 69 0 0 0 1 1 1 1 0 0 3 3 6 1 0 6 10000 10000 1000000 1000000	1 0 20 2 480402900

Note

In the first test case, there exists only one such array: an array consisting of 69 -2s.

In the second test case, the sum of all its elements is $(-2) + (-1) + 1 + 2 = 0$, so there are no such arrays.