A. String

```
Input file:     standard input
Output file:    standard output
Time limit:     1 second
Memory limit:   512 megabytes
```

You are given a string s of length n consisting of 0 and/or 1. In one operation, you can select a non-empty subsequence t from s such that any two adjacent characters in t are different. Then, you flip each character of t (0 becomes 1 and 1 becomes 0). For example, if $s = \underline{00}1\underline{01}$ and $t = s_1 s_3 s_4 s_5 = 0101$, after the operation, s becomes $\underline{1}0\underline{010}$.

Calculate the minimum number of operations required to change all characters in s to 0.

Recall that for a string $s = s_1 s_2 \ldots s_n$, any string $t = s_{i_1} s_{i_2} \ldots s_{i_k}$ ($k \geq 1$) where $1 \leq i_1 < i_2 < \ldots < i_k \leq n$ is a subsequence of s.

Input

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

The only line of each test case contains the string s ($1 \leq |s| \leq 50$), where $|s|$ represents the length of s.

Output

For each test case, output the minimum number of operations required to change all characters in s to 0.

Standard Input	Standard Output
5 1 000 1001 10101 01100101011101	1 0 2 3 8

Note

In the first test case, you can flip s_1. Then s becomes 0, so the answer is 1.

In the fourth test case, you can perform the following three operations in order:

1. Flip $s_1 s_2 s_3 s_4 s_5$. Then s becomes $\underline{01010}$.
2. Flip $s_2 s_3 s_4$. Then s becomes $0\underline{010}0$.
3. Flip s_3. Then s becomes $00\underline{0}00$.

It can be shown that you can not change all characters in s to 0 in less than three operations, so the answer is 3.

,

B. Clockwork

Input file: standard input
Output file: standard output
Time limit: 1.5 seconds
Memory limit: 512 megabytes

You have a sequence of n time clocks arranged in a line, where the initial time on the i-th clock is a_i. In each second, the following happens in order:

- Each clock's time decreases by 1. If any clock's time reaches 0, you lose immediately.
- You can choose to move to an adjacent clock or stay at the clock you are currently on.
- You can reset the time of the clock you are on back to its initial value a_i.

Note that the above events happen in order. If the time of a clock reaches 0 in a certain second, even if you can move to this clock and reset its time during that second, you will still lose.

You can start from any clock. Determine if it is possible to continue this process indefinitely without losing.

Input

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

For each test case, the first line contains a single integer n ($2 \le n \le 5 \cdot 10^5$) — the number of time clocks.

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$) — the initial times set on the clocks.

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

Output

For each test case, print "YES" (without quotes) if it is possible to continue this process indefinitely, or "NO" (without quotes) otherwise.

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

Standard Input	Standard Output
5 2 4 10 2 2 2 3 4 10 5 3 5 3 5 5 12 13 25 17 30	YES NO NO YES YES

Note

In the first test case, you can move back and forth between the two clocks, resetting them repeatedly.

In the third test case, assuming that you start from clock 1 and follow the strategy below:

Initially, $a = [4, 10, 5]$.

1. a becomes $[3, 9, 4]$. You move to clock 2 and reset its time, resulting in $a = [3, 10, 4]$.
2. a becomes $[2, 9, 3]$. You move to clock 3 and reset its time, resulting in $a = [2, 9, 5]$.
3. a becomes $[1, 8, 4]$. You move to clock 2 and reset its time, resulting in $a = [1, 10, 4]$.
4. a becomes $[0, 9, 3]$. You move to clock 1, but you lose because a_1 reaches 0.

It can be proven that no other strategy allows you to continue this process indefinitely.

,

C. Cirno and Operations

Input file: standard input
Output file: standard output
Time limit: 2 seconds
Memory limit: 512 megabytes

Cirno has a sequence a of length n. She can perform either of the following two operations for any (possibly, zero) times **unless** the current length of a is 1:

- Reverse the sequence. Formally, $[a_1, a_2, \ldots, a_n]$ becomes $[a_n, a_{n-1}, \ldots, a_1]$ after the operation.
- Replace the sequence with its difference sequence. Formally, $[a_1, a_2, \ldots, a_n]$ becomes $[a_2 - a_1, a_3 - a_2, \ldots, a_n - a_{n-1}]$ after the operation.

Find the maximum possible sum of elements of a after all operations.

Input

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

The first line of each test case contains a single integer n ($1 \leq n \leq 50$) — the length of sequence a.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($|a_i| \leq 1000$) — the sequence a.

Output

For each test case, print an integer representing the maximum possible sum.

Standard Input	Standard Output
5 1 -1000 2 5 -3 2 1000 1 9 9 7 9 -9 9 -8 7 -8 9 11 678 201 340 444 453 922 128 987 127 752 0	-1000 8 1001 2056 269891

Note

In the first test case, Cirno can not perform any operation, so the answer is -1000.

In the second test case, Cirno firstly reverses the sequence, then replaces the sequence with its difference sequence: $[5, -3] \to [-3, 5] \to [8]$. It can be proven that this maximizes the sum, so the answer is 8.

In the third test case, Cirno can choose not to operate, so the answer is 1001.

,

D. Balanced Tree

Input file: standard input
Output file: standard output
Time limit: 3 seconds
Memory limit: 512 megabytes

You are given a tree* with n nodes and values l_i, r_i for each node. You can choose an initial value a_i satisfying $l_i \le a_i \le r_i$ for the i-th node. A tree is *balanced* if all node values are equal, and the value of a balanced tree is defined as the value of any node.

In one operation, you can choose two nodes u and v, and increase the values of all nodes in the subtree† of node v by 1 while considering u as the root of the entire tree. Note that u may be equal to v.

Your goal is to perform a series of operations so that the tree becomes **balanced**. Find the minimum possible value of the tree after performing these operations. Note that you **don't** need to minimize the number of operations.

*A tree is a connected graph without cycles.

†Node w is considered in the subtree of node v if any path from root u to w must go through v.

Input

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

The first line of each test case contains one integer n ($1 \le n \le 2 \cdot 10^5$) — the number of nodes in the tree.

Then n lines follow. The i-th line contains two integers l_i, r_i ($0 \le l_i \le r_i \le 10^9$) — the constraint of the value of the i-th node.

The next $n-1$ lines contain the edges of the tree. The i-th line contains two integers u_i, v_i ($1 \le u_i, v_i \le n$, $u_i \ne v_i$) — an edge connecting u_i and v_i. It is guaranteed that the given edges form a tree.

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 value that all a_i can be made equal to after performing the operations. It can be shown that the answer always exists.

Standard Input	Standard Output
6 4 0 11 6 6 0 0 5 5 2 1 3 1 4 3 7 1 1 0 5 0 5	11 3 3 5 3000000000 98

```
2 2
2 2
2 2
2 2
1 2
1 3
2 4
2 5
3 6
3 7
4
1 1
1 1
1 1
0 0
1 4
2 4
3 4
7
0 20
0 20
0 20
0 20
3 3
4 4
5 5
1 2
1 3
1 4
2 5
3 6
4 7
5
1000000000 1000000000
0 0
1000000000 1000000000
0 0
1000000000 1000000000
3 2
2 1
1 4
4 5
6
21 88
57 81
98 99
61 76
15 50
23 67
```

2 1 3 2 4 3 5 3 6 4	

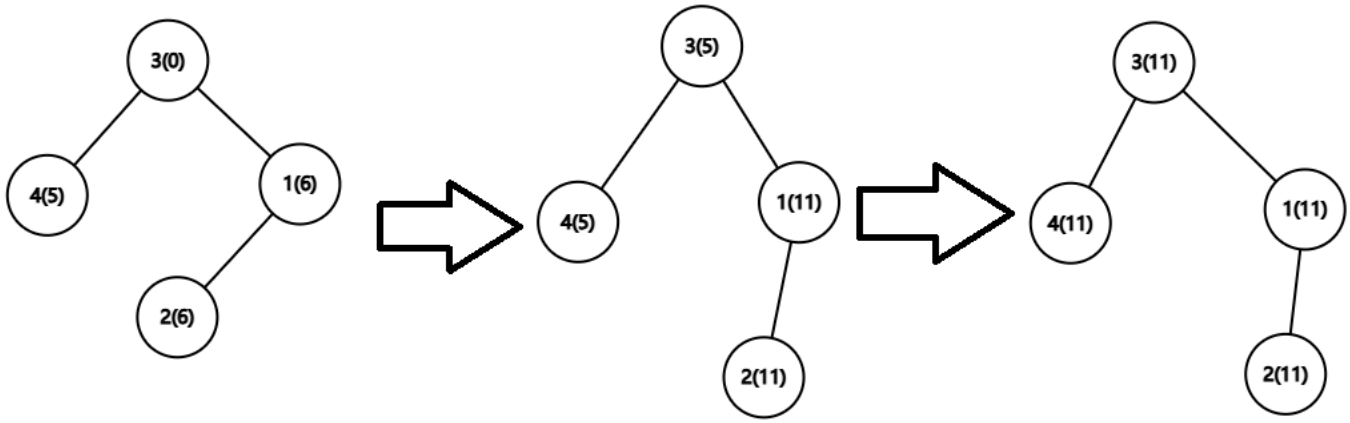
Note

In the first test case, you can choose $a = [6, 6, 0, 5]$.

You can perform the following operations to make all a_i equal:

1. Choose $u = 4$, $v = 3$ and perform the operation 5 times.
2. Choose $u = 1$, $v = 3$ and perform the operation 6 times.

The complete process is shown as follows (where the numbers inside the parentheses are elements of a):



It can be proven that this is the optimal solution.

,

E1. The Game (Easy Version)

Input file: standard input
Output file: standard output
Time limit: 4 seconds
Memory limit: 512 megabytes

This is the easy version of the problem. The difference between the versions is that in this version, you only need to find one of the possible nodes Cirno may choose. You can hack only if you solved all versions of this problem.

Cirno and Daiyousei are playing a game with a tree* of n nodes, rooted at node 1. The value of the i-th node is w_i. They take turns to play the game; Cirno goes first.

In each turn, assuming the opponent chooses j in the last turn, the player can choose any remaining node i satisfying $w_i > w_j$ and delete the subtree† of node i. In particular, Cirno can choose any node and delete its subtree in the first turn.

The first player who **can not** operate **wins**, and they all hope to win. Find **one of the possible nodes Cirno may choose** so that she wins if both of them play optimally.

*A tree is a connected graph without cycles.

†Node u is considered in the subtree of node i if any path from 1 to u must go through i.

Input

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

The first line of each test case contains one integer n ($1 \le n \le 4 \cdot 10^5$) — the number of nodes in the tree.

The second line contains n integers $w_1, w_2, \ldots, w_n$ ($1 \le w_i \le n$) — the value of each node.

The next $n - 1$ lines contain the edges of the tree. The i-th line contains two integers u_i, v_i ($1 \le u_i, v_i \le n$, $u_i \ne v_i$) — an edge connecting u_i and v_i. It is guaranteed that the given edges form a tree.

It is guaranteed that the sum of n over all test cases does not exceed $4 \cdot 10^5$.

Output

For each test case, print one line.

If Cirno wins the game, print any possible node she may choose in the first turn.

Otherwise, print "0" (without quotes).

Standard Input	Standard Output
5 4 2 2 4 3 1 2 1 3 2 4 5 1 2 3 4 5 1 2	2 0 2 2 10

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

Note

In the first test case:

1. If Cirno chooses 1 or 3 in the first turn, Daiyousei cannot make a move, so Daiyousei wins.
2. If Cirno chooses 2 or 4 in the first turn, Daiyousei can only choose 3, but after it Cirno cannot make a move, so Cirno wins.

Therefore, all possible nodes Cirno may choose are 2 and 4.

In the second test case, regardless of which node Cirno chooses, Daiyousei cannot make a move, so Daiyousei wins.

In the third and fourth test case, the only possible node Cirno may choose in the first turn is 2.

In the fifth test case, all possible nodes Cirno may choose in the first turn are $3, 4, 6, 7$ and 10.

,

E2. The Game (Hard Version)

Input file: standard input
Output file: standard output
Time limit: 6 seconds
Memory limit: 512 megabytes

This is the hard version of the problem. The difference between the versions is that in this version, you need to find all possible nodes Cirno may choose. You can hack only if you solved all versions of this problem.

Cirno and Daiyousei are playing a game with a tree* of n nodes, rooted at node 1. The value of the i-th node is w_i. They take turns to play the game; Cirno goes first.

In each turn, assuming the opponent chooses j in the last turn, the player can choose any remaining node i satisfying $w_i > w_j$ and delete the subtree† of node i. In particular, Cirno can choose any node and delete its subtree in the first turn.

The first player who **can not** operate **wins**, and they all hope to win. Find **all possible nodes Cirno may choose in the first turn** so that she wins if both of them play optimally.

*A tree is a connected graph without cycles.

†Node u is considered in the subtree of node i if any path from 1 to u must go through i.

Input

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

The first line of each test case contains one integer n ($1 \leq n \leq 4 \cdot 10^5$) — the number of nodes in the tree.

The second line contains n integers $w_1, w_2, \ldots, w_n$ ($1 \leq w_i \leq n$) — the value of each node.

The next $n - 1$ lines contain the edges of the tree. The i-th line contains two integers u_i, v_i ($1 \leq u_i, v_i \leq n$, $u_i \neq v_i$) — an edge connecting u_i and v_i. It is guaranteed that the given edges form a tree.

It is guaranteed that the sum of n over all test cases does not exceed $4 \cdot 10^5$.

Output

For each test case, print one line.

If Cirno wins the game, print several integers. The first integer k represents the number of possible nodes she may choose in the first turn. The other k integers are all possible nodes in increasing order.

Otherwise, print "0" (without quotes).

Standard Input	Standard Output
5 4 2 2 4 3 1 2 1 3 2 4 5 1 2 3 4 5	2 2 4 0 1 2 1 2 5 3 4 6 7 10

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

Note

In the first test case:

1. If Cirno chooses 1 or 3 in the first turn, Daiyousei cannot make a move, so Daiyousei wins.
2. If Cirno chooses 2 or 4 in the first turn, Daiyousei can only choose 3, but after it Cirno cannot make a move, so Cirno wins.

Therefore, all possible nodes Cirno may choose are 2 and 4.

In the second test case, regardless of which node Cirno chooses, Daiyousei cannot make a move, so Daiyousei wins.

In the third and fourth test case, the only possible node Cirno may choose in the first turn is 2.

In the fifth test case, all possible nodes Cirno may choose in the first turn are $3, 4, 6, 7$ and 10.

,

F. Traveling Salescat

Input file: standard input
Output file: standard output
Time limit: 2 seconds
Memory limit: 512 megabytes

You are a cat selling fun algorithm problems. Today, you want to recommend your fun algorithm problems to k cities.

There are a total of n cities, each with two parameters a_i and b_i. Between any two cities i, j ($i \neq j$), there is a bidirectional road with a length of $\max(a_i + b_j, b_i + a_j)$. The cost of a path is defined as the total length of roads between every two adjacent cities along the path.

For $k = 2, 3, \ldots, n$, find the minimum cost among all simple paths containing exactly k **distinct** cities.

Input

The first line of input contains a single integer t ($1 \leq t \leq 1500$) — the number of test cases.

For each test case, the first line contains a single integer n ($2 \leq n \leq 3 \cdot 10^3$) — the number of cities.

Then n lines follow, the i-th line contains two integers a_i, b_i ($0 \leq a_i, b_i \leq 10^9$) — the parameters of city i.

It is guaranteed that the sum of n^2 over all test cases does not exceed $9 \cdot 10^6$.

Output

For each test case, print $n - 1$ integers in one line. The i-th integer represents the minimum cost when $k = i + 1$.

Standard Input	Standard Output
3 3 0 2 2 1 3 3 5 2 7 7 5 6 3 1 8 7 5 8 899167687 609615846 851467150 45726720 931502759 23784096 918190644 196992738 142090421 475722765 409556751 726971942 513558832 998277529 294328304 434714258	4 9 10 22 34 46 770051069 1655330585 2931719265 3918741472 5033924854 6425541981 7934325514

Note

In the first test case:

- For $k = 2$, the optimal path is $1 \to 2$ with a cost of $\max(0 + 1, 2 + 2) = 4$.
- For $k = 3$, the optimal path is $2 \to 1 \to 3$ with a cost of $\max(0 + 1, 2 + 2) + \max(0 + 3, 3 + 2) = 4 + 5 = 9$.

In the second test case:

- For $k = 2$, the optimal path is $1 \to 4$.
- For $k = 3$, the optimal path is $2 \to 3 \to 5$.
- For $k = 4$, the optimal path is $4 \to 1 \to 3 \to 5$.
- For $k = 5$, the optimal path is $5 \to 2 \to 3 \to 1 \to 4$.

,

G. Permutation Factory

Input file: standard input
Output file: standard output
Time limit: 2 seconds
Memory limit: 512 megabytes

You are given two permutations $p_1, p_2, \ldots, p_n$ and $q_1, q_2, \ldots, q_n$ of length n. In one operation, you can select two integers $1 \leq i, j \leq n, i \neq j$ and swap p_i and p_j. The cost of the operation is $\min(|i - j|, |p_i - p_j|)$.

Find the minimum cost to make $p_i = q_i$ hold for all $1 \leq i \leq n$ and output a sequence of operations to achieve the minimum cost.

A permutation of length n is an array consisting of n distinct integers from 1 to n in arbitrary order. For example, $[2, 3, 1, 5, 4]$ is a permutation, but $[1, 2, 2]$ is not a permutation (2 appears twice in the array), and $[1, 3, 4]$ is also not a permutation ($n = 3$ but there is 4 in the array).

Input

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

The first line of each test case contains one integer n ($2 \leq n \leq 100$) — the length of permutations p and q.

The second line contains n integers $p_1, p_2, \ldots, p_n$ ($1 \leq p_i \leq n$) — the permutation p. It is guaranteed that $p_1, p_2, \ldots, p_n$ is a permutation of $1, 2, \ldots, n$.

The third line contains n integers $q_1, q_2, \ldots, q_n$ ($1 \leq q_i \leq n$) — the permutation q. It is guaranteed that $q_1, q_2, \ldots, q_n$ is a permutation of $1, 2, \ldots, n$.

It is guaranteed that the sum of n^3 over all test cases does not exceed 10^6.

Output

For each test case, output the total number of operations k ($0 \leq k \leq n^2$) on the first line. Then output k lines, each containing two integers i, j ($1 \leq i, j \leq n, i \neq j$) representing an operation to swap p_i and p_j in order.

It can be shown that no optimal operation sequence has a length greater than n^2.

Standard Input	Standard Output
4 2 2 1 2 1 3 1 2 3 3 2 1 4 2 1 4 3 4 2 3 1 5 1 4 3 2 5 5 2 3 4 1	0 1 1 3 3 1 4 2 4 1 3 4 1 2 4 5 2 5 1 4

Note

In the second test case, you can swap p_1, p_3 costing $\min(|1-3|, |1-3|) = 2$. Then p equals q with a cost of 2.

In the third test case, you can perform the following operations:

Initially, $p = [2, 1, 4, 3]$.

1. Swap p_1, p_4 costing $\min(|1-4|, |2-3|) = 1$, resulting in $p = [3, 1, 4, 2]$.
2. Swap p_2, p_4 costing $\min(|2-4|, |1-2|) = 1$, resulting in $p = [3, 2, 4, 1]$.
3. Swap p_1, p_3 costing $\min(|1-3|, |3-4|) = 1$. Then p equals q with a cost of 3.

,

H. Galaxy Generator

Input file: standard input
Output file: standard output
Time limit: 4 seconds
Memory limit: 512 megabytes

In a two-dimensional universe, a star can be represented by a point (x, y) on a two-dimensional plane. Two stars are directly connected if and only if their x or y coordinates are the same, and there are no other stars on the line segment between them. Define a galaxy as a connected component composed of stars connected directly or indirectly (through other stars).

For a set of stars, its value is defined as the minimum number of galaxies that can be obtained after performing the following operation for any (possibly, zero) times: in each operation, you can select a point (x, y) without stars. If a star can be directly connected to at least 3 stars after creating it here, then you create a star here.

You are given a $n \times n$ matrix a consisting of 0 and 1 describing a set S of stars. There is a star at (x, y) if and only if $a_{x,y} = 1$. Calculate the sum, modulo $10^9 + 7$, of the values of all non-empty subsets of S.

Input

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

For each test case, the first line contains a single integer n ($1 \leq n \leq 14$) — the size of matrix a.

Then n lines follow; the i-th line contains a string a_i of length n — the i-th row of matrix a.

It is guaranteed that the sum of 2^n over all test cases does not exceed 2^{14}.

Output

For each test case, output the sum, modulo $10^9 + 7$, of the values of all non-empty subsets of S.

Standard Input	Standard Output
8 1 0 2 01 10 3 010 000 101 4 0110 1001 1001 0110 11 11111110111 10000010010 10111010011	0 4 9 355 593092633 438667113 922743932 155

10111010011 10111010001 10000010000 11111110101 00000000111 11011010011 10010101100 11101010100 11 11011111110 10010000010 00010111010 10010111010 01010111010 11010000010 01011111110 11000000000 01010000010 01000111100 00000001010 11 11010101001 11001010100 00000000110 11111110010 10000010010 10111010110 10111010111 10111010010 10000010110 11111110100 00000000000 3 111 100 111	

Note

In the first test case, S is empty. S has no non-empty subsets. So the answer is 0.

In the second test case, $S = \{(1, 2), (2, 1)\}$. S has 3 non-empty subsets.

- $\{(1, 2)\}$ and $\{(2, 1)\}$ — there is only one star in the set, forming 1 galaxy.
- $\{(1, 2), (2, 1)\}$ — two stars in the set are not connected, forming 2 galaxies.

So the answer is $1 + 1 + 2 = 4$.

In the third test case, $S = \{(1, 2), (3, 1), (3, 3)\}$. S has 7 non-empty subsets.

- $\{(1, 2)\}$, $\{(3, 1)\}$, and $\{(3, 3)\}$ — there is only one star in the set, forming 1 galaxy.
- $\{(1, 2), (3, 1)\}$ and $\{(1, 2), (3, 3)\}$ — two stars in the set are not connected, forming 2 galaxies.

- $\{(3,1),(3,3)\}$ — two stars in the set are connected, forming 1 galaxy.
- $\{(1,2),(3,1),(3,3)\}$ — initially, star $(1,2)$ is not in the galaxy formed by $(3,1)$ and $(3,3)$. You can make an operation creating a star at $(3,2)$ connecting to these three stars, forming 1 galaxy.

So the answer is $1+1+1+2+2+1+1=9$.