Codeforces Round #661 (Div. 3)

A. Remove Smallest

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given the array a consisting of n positive (greater than zero) integers.

In one move, you can choose two indices i and j ($i \neq j$) such that the absolute difference between a_i and a_j is no more than one ($|a_i - a_j| \leq 1$) and remove the smallest of these two elements. If two elements are equal, you can remove any of them (but exactly one).

Your task is to find if it is possible to obtain the array consisting of **only one element** using several (possibly, zero) such moves or not.

You have to answer t independent test cases.

Input

The first line of the input contains one integer t ($1 \leq t \leq 1000$) — the number of test cases. Then t test cases follow.

The first line of the test case contains one integer n ($1 \leq n \leq 50$) — the length of a. The second line of the test case contains n integers $a_1, a_2, \ldots, a_n$ ($1 \leq a_i \leq 100$), where a_i is the i-th element of a.

Output

For each test case, print the answer: "YES" if it is possible to obtain the array consisting of **only one element** using several (possibly, zero) moves described in the problem statement, or "NO" otherwise.

Example

input
5 3 1 2 2 4 5 5 5 5 3 1 2 4 4 1 3 4 4 1 100
output
YES YES NO NO YES

Note

In the first test case of the example, we can perform the following sequence of moves:

- choose $i = 1$ and $j = 3$ and remove a_i (so a becomes $[2; 2]$);
- choose $i = 1$ and $j = 2$ and remove a_j (so a becomes $[2]$).

In the second test case of the example, we can choose any possible i and j any move and it doesn't matter which element we remove.

In the third test case of the example, there is no way to get rid of 2 and 4.

B. Gifts Fixing

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

You have n gifts and you want to give all of them to children. Of course, you don't want to offend anyone, so all gifts should be equal between each other. The i-th gift consists of a_i candies and b_i oranges.

During one move, you can choose some gift $1 \leq i \leq n$ and do one of the following operations:

- eat exactly **one candy** from this gift (decrease a_i by one);
- eat exactly **one orange** from this gift (decrease b_i by one);
- eat exactly **one candy** and exactly **one orange** from this gift (decrease both a_i and b_i by one).

Of course, you can not eat a candy or orange if it's not present in the gift (so neither a_i nor b_i can become less than zero).

As said above, all gifts should be equal. This means that after some sequence of moves the following two conditions should be satisfied: $a_1 = a_2 = \dots = a_n$ and $b_1 = b_2 = \dots = b_n$ (and a_i equals b_i is **not necessary**).

Your task is to find the **minimum** number of moves required to equalize all the given gifts.

You have to answer t independent test cases.

Input
The first line of the input contains one integer t ($1 \le t \le 1000$) — the number of test cases. Then t test cases follow.

The first line of the test case contains one integer n ($1 \le n \le 50$) — the number of gifts. The second line of the test case contains n integers $a_1, a_2, \dots, a_n$ ($1 \le a_i \le 10^9$), where a_i is the number of candies in the i-th gift. The third line of the test case contains n integers $b_1, b_2, \dots, b_n$ ($1 \le b_i \le 10^9$), where b_i is the number of oranges in the i-th gift.

Output
For each test case, print one integer: the **minimum** number of moves required to equalize all the given gifts.

Example

input
5 3 3 5 6 3 2 3 5 1 2 3 4 5 5 4 3 2 1 3 1 1 1 2 2 2 6 1 1000000000 1000000000 1000000000 1000000000 1000000000 1 1 1 1 1 1 3 10 12 8 7 5 4
output
6 16 0 4999999995 7

Note
In the first test case of the example, we can perform the following sequence of moves:

- choose the first gift and eat one orange from it, so $a = [3, 5, 6]$ and $b = [2, 2, 3]$;
- choose the second gift and eat one candy from it, so $a = [3, 4, 6]$ and $b = [2, 2, 3]$;
- choose the second gift and eat one candy from it, so $a = [3, 3, 6]$ and $b = [2, 2, 3]$;
- choose the third gift and eat one candy and one orange from it, so $a = [3, 3, 5]$ and $b = [2, 2, 2]$;
- choose the third gift and eat one candy from it, so $a = [3, 3, 4]$ and $b = [2, 2, 2]$;
- choose the third gift and eat one candy from it, so $a = [3, 3, 3]$ and $b = [2, 2, 2]$.

C. Boats Competition

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

There are n people who want to participate in a boat competition. The weight of the i-th participant is w_i. Only teams consisting of **two** people can participate in this competition. As an organizer, you think that it's fair to allow only teams with **the same total weight**.

So, if there are k teams (a_1, b_1), (a_2, b_2), …, (a_k, b_k), where a_i is the weight of the first participant of the i-th team and b_i is the weight of the second participant of the i-th team, then the condition $a_1 + b_1 = a_2 + b_2 = \dots = a_k + b_k = s$, where s is the total weight of **each** team, should be satisfied.

Your task is to choose such s that the number of teams people can create is the **maximum** possible. Note that each participant can be in **no more than one** team.

You have to answer t independent test cases.

Input
The first line of the input contains one integer t ($1 \le t \le 1000$) — the number of test cases. Then t test cases follow.

The first line of the test case contains one integer n ($1 \le n \le 50$) — the number of participants. The second line of the test case contains n integers $w_1, w_2, \ldots, w_n$ ($1 \le w_i \le n$), where w_i is the weight of the i-th participant.

Output
For each test case, print one integer k: the **maximum** number of teams people can compose with the total weight s, if you choose s optimally.

Example

input
5 5 1 2 3 4 5 8 6 6 6 6 6 6 8 8 8 1 2 2 1 2 1 1 2 3 1 3 3 6 1 1 3 4 2 2
output
2 3 4 1 2

Note
In the first test case of the example, we can reach the optimal answer for $s = 6$. Then the first boat is used by participants 1 and 5 and the second boat is used by participants 2 and 4 (indices are the same as weights).

In the second test case of the example, we can reach the optimal answer for $s = 12$. Then first 6 participants can form 3 pairs.

In the third test case of the example, we can reach the optimal answer for $s = 3$. The answer is 4 because we have 4 participants with weight 1 and 4 participants with weight 2.

In the fourth test case of the example, we can reach the optimal answer for $s = 4$ or $s = 6$.

In the fifth test case of the example, we can reach the optimal answer for $s = 3$. Note that participant with weight 3 can't use the boat because there is no suitable pair for him in the list.

D. Binary String To Subsequences

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given a binary string s consisting of n zeros and ones.

Your task is to divide the given string into the **minimum** number of **subsequences** in such a way that *each character* of the string belongs to exactly *one subsequence* and each subsequence looks like "`010101 ...`" or "`101010 ...`" (i.e. the subsequence should not contain two adjacent zeros or ones).

Recall that a subsequence is a sequence that can be derived from the given sequence by deleting zero or more elements without changing the order of the remaining elements. For example, subsequences of "`1011101`" are "`0`", "`1`", "`11111`", "`0111`", "`101`", "`1001`", but not "`000`", "`101010`" and "`11100`".

You have to answer t independent test cases.

Input
The first line of the input contains one integer t ($1 \le t \le 2 \cdot 10^4$) — the number of test cases. Then t test cases follow.

The first line of the test case contains one integer n ($1 \le n \le 2 \cdot 10^5$) — the length of s. The second line of the test case contains n characters '`0`' and '`1`' — the string s.

It is guaranteed that the sum of n does not exceed $2 \cdot 10^5$ ($\sum n \le 2 \cdot 10^5$).

Output
For each test case, print the answer: in the first line print one integer k ($1 \le k \le n$) — the minimum number of subsequences you can divide the string s to. In the second line print n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le k$), where a_i is the number of subsequence the i-th character of s belongs to.

If there are several answers, you can print any.

Example

input
4 4 0011 6 111111 5 10101 8 01010000

output
2 1 2 2 1 6 1 2 3 4 5 6 1 1 1 1 1 1 4 1 1 1 1 1 2 3 4

E1. Weights Division (easy version)

time limit per test: 3 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Easy and hard versions are actually different problems, so we advise you to read both statements carefully.

You are given a weighted rooted tree, vertex 1 is the root of this tree.

A tree is a connected graph without cycles. A rooted tree has a special vertex called the root. A parent of a vertex v is the last different from v vertex on the path from the root to the vertex v. Children of vertex v are all vertices for which v is the parent. A vertex is a leaf if it has no children. The weighted tree is such a tree that each edge of this tree has some weight.

The weight of the path is the sum of edges weights on this path. The weight of the path from the vertex to itself is 0.

You can make a sequence of zero or more moves. On each move, you select an edge and divide its weight by 2 rounding down. More formally, during one move, you choose some edge i and divide its weight by 2 rounding down ($w_i := \left\lfloor \frac{w_i}{2} \right\rfloor$).

Your task is to find the minimum number of **moves** required to make the **sum of weights of paths** from the root to each leaf at most S. In other words, if $w(i, j)$ is the weight of the path from the vertex i to the vertex j, then you have to make $\sum\limits_{v \in leaves} w(root, v) \le S$, where $leaves$ is the list of all leaves.

You have to answer t independent test cases.

Input

The first line of the input contains one integer t ($1 \le t \le 2 \cdot 10^4$) — the number of test cases. Then t test cases follow.

The first line of the test case contains two integers n and S ($2 \le n \le 10^5; 1 \le S \le 10^{16}$) — the number of vertices in the tree and the maximum possible sum of weights you have to obtain. The next $n-1$ lines describe edges of the tree. The edge i is described as three integers v_i, u_i and w_i ($1 \le v_i, u_i \le n; 1 \le w_i \le 10^6$), where v_i and u_i are vertices the edge i connects and w_i is the weight of this edge.

It is guaranteed that the sum of n does not exceed 10^5 ($\sum n \le 10^5$).

Output

For each test case, print the answer: the minimum number of **moves** required to make the sum of weights of paths from the root to each leaf at most S.

Example

input
3 3 20 2 1 8 3 1 7 5 50 1 3 100 1 5 10 2 3 123 5 4 55 2 100 1 2 409

output
0 8 3

E2. Weights Division (hard version)

time limit per test: 3 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Easy and hard versions are actually different problems, so we advise you to read both statements carefully.

You are given a weighted rooted tree, vertex 1 is the root of this tree. **Also, each edge has its own cost**.

A tree is a connected graph without cycles. A rooted tree has a special vertex called the root. A parent of a vertex v is the last different from v vertex on the path from the root to the vertex v. Children of vertex v are all vertices for which v is the parent. A vertex is a leaf if it has no children. The weighted tree is such a tree that each edge of this tree has some weight.

The weight of the path is the sum of edges weights on this path. The weight of the path from the vertex to itself is 0.

You can make a sequence of zero or more moves. On each move, you select an edge and divide its weight by 2 rounding down. More formally, during one move, you choose some edge i and divide its weight by 2 rounding down ($w_i := \left\lfloor \frac{w_i}{2} \right\rfloor$).

Each edge i has an associated cost c_i which is either 1 or 2 coins. Each move with edge i costs c_i coins.

Your task is to find the minimum total **cost** to make the **sum of weights of paths** from the root to each leaf at most S. In other words, if $w(i, j)$ is the weight of the path from the vertex i to the vertex j, then you have to make $\sum\limits_{v \in leaves} w(root, v) \le S$, where $leaves$ is the list of all leaves.

You have to answer t independent test cases.

Input

The first line of the input contains one integer t ($1 \le t \le 2 \cdot 10^4$) — the number of test cases. Then t test cases follow.

The first line of the test case contains two integers n and S ($2 \le n \le 10^5; 1 \le S \le 10^{16}$) — the number of vertices in the tree and the maximum possible sum of weights you have to obtain. The next $n-1$ lines describe edges of the tree. The edge i is described as four integers v_i, u_i, w_i and c_i ($1 \le v_i, u_i \le n; 1 \le w_i \le 10^6; 1 \le c_i \le 2$), where v_i and u_i are vertices the edge i connects, w_i is the weight of this edge and c_i is the cost of this edge.

It is guaranteed that the sum of n does not exceed 10^5 ($\sum n \le 10^5$).

Output

For each test case, print the answer: the minimum total **cost** required to make the sum of weights paths from the root to each leaf at most S.

Example

input
4 4 18 2 1 9 2 3 2 4 1 4 1 1 2 3 20 2 1 8 1 3 1 7 2 5 50 1 3 100 1 1 5 10 2 2 3 123 2 5 4 55 1 2 100 1 2 409 2
output
0 0 11 6

F. Yet Another Segments Subset

time limit per test: 3 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given n segments on a coordinate axis OX. The i-th segment has borders $[l_i; r_i]$. All points x, for which $l_i \le x \le r_i$ holds, belong to the i-th segment.

Your task is to choose the **maximum** by size (the number of segments) subset of the given set of segments such that each pair of segments in this subset either non-intersecting or one of them lies inside the other one.

Two segments $[l_i; r_i]$ and $[l_j; r_j]$ are non-intersecting if they have **no common points**. For example, segments $[1; 2]$ and $[3; 4]$, $[1; 3]$ and $[5; 5]$ are non-intersecting, while segments $[1; 2]$ and $[2; 3]$, $[1; 2]$ and $[2; 2]$ are intersecting.

The segment $[l_i; r_i]$ lies inside the segment $[l_j; r_j]$ if $l_j \le l_i$ and $r_i \le r_j$. For example, segments $[2; 2]$, $[2, 3]$, $[3; 4]$ and $[2; 4]$ lie inside the segment $[2; 4]$, while $[2; 5]$ and $[1; 4]$ are not.

You have to answer t independent test cases.

Input

The first line of the input contains one integer t ($1 \le t \le 1000$) — the number of test cases. Then t test cases follow.

The first line of the test case contains one integer n ($1 \le n \le 3000$) — the number of segments. The next n lines describe segments. The i-th segment is given as two integers l_i and r_i ($1 \le l_i \le r_i \le 2 \cdot 10^5$), where l_i is the left border of the i-th segment and r_i is the right border of the i-th segment.

Additional constraint on the input: there are **no duplicates** in the list of segments.

It is guaranteed that the sum of n does not exceed 3000 ($\sum n \le 3000$).

Output

For each test case, print the answer: the **maximum** possible size of the subset of the given set of segments such that each pair of segments in this subset either non-intersecting or one of them lies inside the other one.

Example

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

output
3 4 2 7