A. Robot Cleaner

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

A robot cleaner is placed on the floor of a rectangle room, surrounded by walls. The floor consists of n rows and m columns. The rows of the floor are numbered from 1 to n from top to bottom, and columns of the floor are numbered from 1 to m from left to right. The cell on the intersection of the r-th row and the c-th column is denoted as (r, c). The initial position of the robot is (r_b, c_b).

In one second, the robot moves by dr rows and dc columns, that is, after one second, the robot moves from the cell (r, c) to $(r + dr, c + dc)$. Initially $dr = 1$, $dc = 1$. If there is a vertical wall (the left or the right walls) in the movement direction, dc is *reflected* before the movement, so the new value of dc is $-dc$. And if there is a horizontal wall (the upper or lower walls), dr is *reflected* before the movement, so the new value of dr is $-dr$.

Each second (including the moment before the robot starts moving), the robot cleans every cell lying in the same row **or** the same column as its position. There is only one dirty cell at (r_d, c_d). The job of the robot is to clean that dirty cell.

Illustration for the first example. The blue arc is the robot. The red star is the target dirty cell. Each second the robot cleans a row and a column, denoted by yellow stripes.

Given the floor size n and m, the robot's initial position (r_b, c_b) and the dirty cell's position (r_d, c_d), find the time for the robot to do its job.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 10^4$). Description of the test cases follows.

A test case consists of only one line, containing six integers n, m, r_b, c_b, r_d, and c_d ($1 \le n, m \le 100$, $1 \le r_b, r_d \le n$, $1 \le c_b, c_d \le m$) — the sizes of the room, the initial position of the robot and the position of the dirt cell.

Output

For each test case, print an integer — the time for the robot to clean the dirty cell. We can show that the robot always cleans the dirty cell eventually.

Example

input
5 10 10 6 1 2 8 10 10 9 9 1 1 9 8 5 6 2 1 6 9 2 2 5 8 2 2 1 1 2 1
output
7 10 9 3 0

Note

In the first example, the floor has the size of 10×10. The initial position of the robot is $(6, 1)$ and the position of the dirty cell is

$(2, 8)$. See the illustration of this example in the problem statement.

In the second example, the floor is the same, but the initial position of the robot is now $(9, 9)$, and the position of the dirty cell is $(1, 1)$. In this example, the robot went straight to the dirty cell and clean it.

In the third example, the floor has the size 9×8. The initial position of the robot is $(5, 6)$, and the position of the dirty cell is $(2, 1)$.

In the fourth example, the floor has the size 6×9. The initial position of the robot is $(2, 2)$ and the position of the dirty cell is $(5, 8)$.

In the last example, the robot was already standing in the same column as the dirty cell, so it can clean the cell right away.

B. Game on Ranges

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

Alice and Bob play the following game. Alice has a set S of disjoint ranges of integers, initially containing only one range $[1, n]$. In one turn, Alice picks a range $[l, r]$ from the set S and asks Bob to pick a number in the range. Bob chooses a number d ($l \le d \le r$). Then Alice removes $[l, r]$ from S and puts into the set S the range $[l, d - 1]$ (if $l \le d - 1$) and the range $[d + 1, r]$ (if $d + 1 \le r$). The game ends when the set S is empty. We can show that the number of turns in each game is exactly n.

After playing the game, Alice remembers all the ranges $[l, r]$ she picked from the set S, but Bob does not remember any of the numbers that he picked. But Bob is smart, and he knows he can find out his numbers d from Alice's ranges, and so he asks you for help with your programming skill.

Given the list of ranges that Alice has picked ($[l, r]$), for each range, help Bob find the number d that Bob has picked.

We can show that there is always a unique way for Bob to choose his number for a list of valid ranges picked by Alice.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 1000$). Description of the test cases follows.

The first line of each test case contains a single integer n ($1 \le n \le 1000$).

Each of the next n lines contains two integers l and r ($1 \le l \le r \le n$), denoting the range $[l, r]$ that Alice picked at some point.

Note that the ranges are given **in no particular order**.

It is guaranteed that the sum of n over all test cases does not exceed 1000, and the ranges for each test case are from a valid game.

Output

For each test case print n lines. Each line should contain three integers l, r, and d, denoting that for Alice's range $[l, r]$ Bob picked the number d.

You can print the lines in **any order**. We can show that the answer is unique.

It is not required to print a new line after each test case. The new lines in the output of the example are for readability only.

Example

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

Note

In the first test case, there is only 1 range $[1, 1]$. There was only one range $[1, 1]$ for Alice to pick, and there was only one number 1 for Bob to pick.

In the second test case, $n = 3$. Initially, the set contains only one range $[1, 3]$.

- Alice picked the range $[1, 3]$. Bob picked the number 1. Then Alice put the range $[2, 3]$ back to the set, which after this turn is the only range in the set.
- Alice picked the range $[2, 3]$. Bob picked the number 3. Then Alice put the range $[2, 2]$ back to the set.
- Alice picked the range $[2, 2]$. Bob picked the number 2. The game ended.

In the fourth test case, the game was played with $n = 5$. Initially, the set contains only one range $[1, 5]$. The game's turn is described in the following table.

Game turn	Alice's picked range	Bob's picked number	The range set after
Before the game start			$\{[1, 5]\}$
1	$[1, 5]$	3	$\{[1, 2], [4, 5]\}$
2	$[1, 2]$	1	$\{[2, 2], [4, 5]\}$

3	$[4, 5]$	5	$\{[2, 2], [4, 4]\}$
4	$[2, 2]$	2	$\{[4, 4]\}$
5	$[4, 4]$	4	$\{\}$ (empty set)

C. Balanced Stone Heaps

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

There are n heaps of stone. The i-th heap has h_i stones. You want to change the number of stones in the heap by performing the following process once:

- You go through the heaps from the 3-rd heap to the n-th heap, in this order.
- Let i be the number of the current heap.
- You can choose a number d ($0 \leq 3 \cdot d \leq h_i$), move d stones from the i-th heap to the $(i-1)$-th heap, and $2 \cdot d$ stones from the i-th heap to the $(i-2)$-th heap.
- So after that h_i is decreased by $3 \cdot d$, h_{i-1} is increased by d, and h_{i-2} is increased by $2 \cdot d$.
- You can choose different or same d for different operations. Some heaps may become empty, but they still count as heaps.

What is the maximum number of stones in the smallest heap after the process?

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \leq t \leq 2 \cdot 10^5$). Description of the test cases follows.

The first line of each test case contains a single integer n ($3 \leq n \leq 2 \cdot 10^5$).

The second lines of each test case contains n integers $h_1, h_2, h_3, \ldots, h_n$ ($1 \leq h_i \leq 10^9$).

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 the maximum number of stones that the smallest heap can contain.

Example

input
4 4 1 2 10 100 4 100 100 100 1 5 5 1 1 1 8 6 1 2 3 4 5 6
output
7 1 1 3

Note

In the first test case, the initial heap sizes are $[1, 2, 10, 100]$. We can move the stones as follows.

- move 3 stones and 6 from the 3-rd heap to the 2-nd and 1 heap respectively. The heap sizes will be $[7, 5, 1, 100]$;
- move 6 stones and 12 stones from the last heap to the 3-rd and 2-nd heap respectively. The heap sizes will be $[7, 17, 7, 82]$.

In the second test case, the last heap is 1, and we can not increase its size.

In the third test case, it is better not to move any stones.

In the last test case, the final achievable configuration of the heaps can be $[3, 5, 3, 4, 3, 3]$.

D. Robot Cleaner Revisit

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

The statement of this problem shares a lot with problem A. The differences are that in this problem, the probability is introduced,

and the constraint is different.

A robot cleaner is placed on the floor of a rectangle room, surrounded by walls. The floor consists of n rows and m columns. The rows of the floor are numbered from 1 to n from top to bottom, and columns of the floor are numbered from 1 to m from left to right. The cell on the intersection of the r-th row and the c-th column is denoted as (r, c). The initial position of the robot is (r_b, c_b).

In one second, the robot moves by dr rows and dc columns, that is, after one second, the robot moves from the cell (r, c) to $(r + dr, c + dc)$. Initially $dr = 1$, $dc = 1$. If there is a vertical wall (the left or the right walls) in the movement direction, dc is *reflected* before the movement, so the new value of dc is $-dc$. And if there is a horizontal wall (the upper or lower walls), dr is *reflected* before the movement, so the new value of dr is $-dr$.

Each second (including the moment before the robot starts moving), the robot cleans every cell lying in the same row **or** the same column as its position. There is only one dirty cell at (r_d, c_d). The job of the robot is to clean that dirty cell.

After a lot of testings in problem A, the robot is now broken. It cleans the floor as described above, but at each second the cleaning operation is performed with probability $\frac{p}{100}$ only, and not performed with probability $1 - \frac{p}{100}$. The cleaning or not cleaning outcomes are independent each second.

Given the floor size n and m, the robot's initial position (r_b, c_b) and the dirty cell's position (r_d, c_d), find the **expected time** for the robot to do its job.

It can be shown that the answer can be expressed as an irreducible fraction $\frac{x}{y}$, where x and y are integers and $y \not\equiv 0 \pmod{10^9 + 7}$. Output the integer equal to $x \cdot y^{-1} \bmod (10^9 + 7)$. In other words, output such an integer a that $0 \le a < 10^9 + 7$ and $a \cdot y \equiv x \pmod{10^9 + 7}$.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 10$). Description of the test cases follows.

A test case consists of only one line, containing n, m, r_b, c_b, r_d, c_d, and p ($4 \le n \cdot m \le 10^5$, $n, m \ge 2$, $1 \le r_b, r_d \le n$, $1 \le c_b, c_d \le m$, $1 \le p \le 99$) — the sizes of the room, the initial position of the robot, the position of the dirt cell and the probability of cleaning in percentage.

Output

For each test case, print a single integer — the expected time for the robot to clean the dirty cell, modulo $10^9 + 7$.

Example

input
6 2 2 1 1 2 1 25 3 3 1 2 2 2 25 10 10 1 1 10 10 75 10 10 10 10 1 1 75 5 5 1 3 2 2 10 97 98 3 5 41 43 50
output
3 3 15 15 332103349 99224487

Note

In the first test case, the robot has the opportunity to clean the dirty cell every second. Using the geometric distribution, we can find out that with the success rate of 25%, the expected number of tries to clear the dirty cell is $\frac{1}{0.25} = 4$. But because the first moment the robot has the opportunity to clean the cell is before the robot starts moving, the answer is 3.

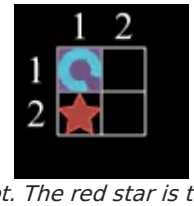


Illustration for the first example. The blue arc is the robot. The red star is the target dirt cell. The purple square is the initial position of the robot. Each second the robot has an opportunity to clean a row and a column, denoted by yellow stripes.

In the second test case, the board size and the position are different, but the robot still has the opportunity to clean the dirty cell every second, and it has the same probability of cleaning. Therefore the answer is the same as in the first example.

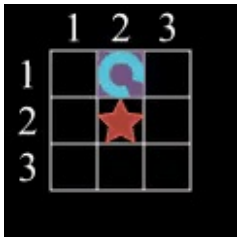


Illustration for the second example.

The third and the fourth case are almost the same. The only difference is that the position of the dirty cell and the robot are swapped. But the movements in both cases are identical, hence the same result.

E. Middle Duplication

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

A binary tree of n nodes is given. Nodes of the tree are numbered from 1 to n and the root is the node 1. Each node can have no child, only one left child, only one right child, or both children. For convenience, let's denote l_u and r_u as the left and the right child of the node u respectively, $l_u = 0$ if u does not have the left child, and $r_u = 0$ if the node u does not have the right child.

Each node has a string label, initially is a single character c_u. Let's define the string representation of the binary tree as the concatenation of the labels of the nodes in the *in-order*. Formally, let $f(u)$ be the string representation of the tree rooted at the node u. $f(u)$ is defined as follows:

$$f(u) = \begin{cases} \texttt{<empty string>}, & \text{if } u = 0; \\ f(l_u) + c_u + f(r_u) & \text{otherwise,} \end{cases}$$

where $+$ denotes the string concatenation operation.

This way, the string representation of the tree is $f(1)$.

For each node, we can *duplicate* its label **at most once**, that is, assign c_u with $c_u + c_u$, but only if u is the root of the tree, or if its parent also has its label duplicated.

You are given the tree and an integer k. What is the lexicographically smallest string representation of the tree, if we can duplicate labels of at most k nodes?

A string a is lexicographically smaller than a string b if and only if one of the following holds:

- a is a prefix of b, but $a \neq b$;
- in the first position where a and b differ, the string a has a letter that appears earlier in the alphabet than the corresponding letter in b.

Input

The first line contains two integers n and k ($1 \leq k \leq n \leq 2 \cdot 10^5$).

The second line contains a string c of n lower-case English letters, where c_i is the initial label of the node i for $1 \leq i \leq n$. Note that the given string c is **not** the initial string representation of the tree.

The i-th of the next n lines contains two integers l_i and r_i ($0 \leq l_i, r_i \leq n$). If the node i does not have the left child, $l_i = 0$, and if the node i does not have the right child, $r_i = 0$.

It is guaranteed that the given input forms a binary tree, rooted at 1.

Output

Print a single line, containing the lexicographically smallest string representation of the tree if at most k nodes have their labels duplicated.

Examples

input
4 3 abab 2 3 0 0 0 4 0 0
output
baaaab

input
8 2 kadracyn 2 5

```
3 4
0 0
0 0
6 8
0 7
0 0
0 0
```

output

```
daarkkcyan
```

input

```
8 3
kdaracyn
2 5
0 3
0 4
0 0
6 8
0 7
0 0
0 0
```

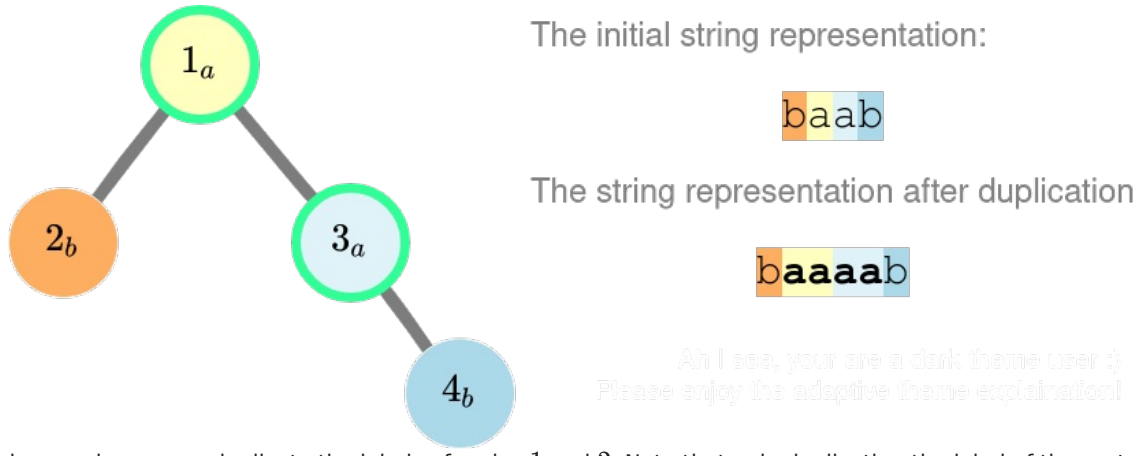
output

```
darkcyan
```

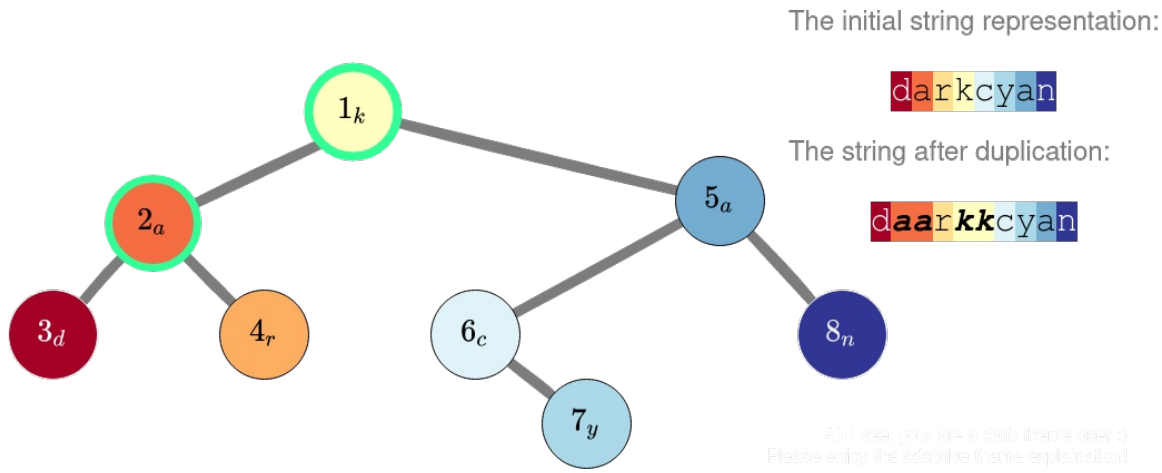
Note

The images below present the tree for the examples. The number in each node is the node number, while the subscripted letter is its label. To the right is the string representation of the tree, with each letter having the same color as the corresponding node.

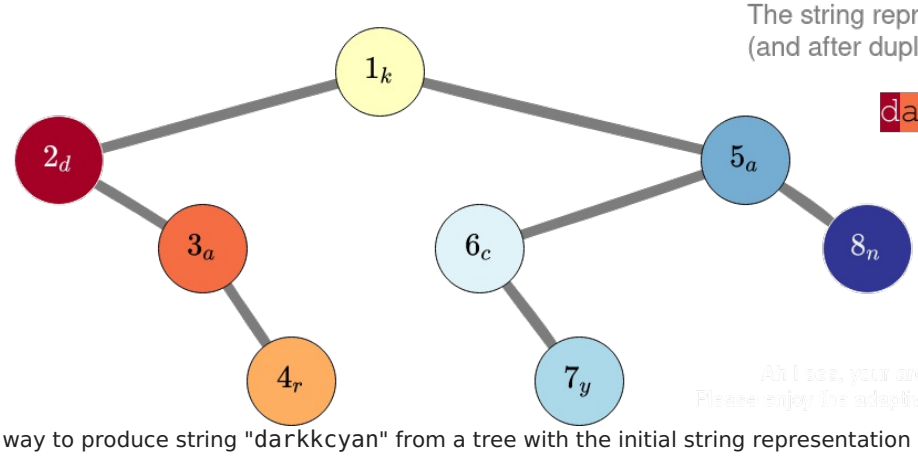
Here is the tree for the first example. Here we duplicated the labels of nodes 1 and 3. We should not duplicate the label of node 2 because it would give us the string "bbaaab", which is lexicographically greater than "baaaab".



In the second example, we can duplicate the labels of nodes 1 and 2. Note that only duplicating the label of the root will produce a worse result than the initial string.



In the third example, we should not duplicate any character at all. Even though we would want to duplicate the label of the node 3, by duplicating it we must also duplicate the label of the node 2, which produces a worse result.



There is no way to produce string "darkkcyan" from a tree with the initial string representation "darkcyan" :(.
