Kickstart Round D 2017

Problem A. Go Sightseeing

Small input 10 points	Solve A-small
Large input 14 points	Solve A-large

Problem

When you travel, you like to spend time sightseeing in as many cities as possible, but sometimes you might not be able to because you need to catch the bus to the next city. To maximize your travel enjoyment, you decide to write a program to optimize your schedule.

You begin at city 1 at time 0 and plan to travel to cities 2 to **N** in ascending order, visiting every city. There is a bus service from every city i to the next city i + 1. The i-th bus service runs on a schedule that is specified by 3 integers: $\mathbf{S_i}$, $\mathbf{F_i}$ and $\mathbf{D_i}$, the start time, frequency and ride duration. Formally, this means that there is a bus leaving from city i at all times $\mathbf{S_i} + x\mathbf{F_i}$, where x is an integer and $x \geq 0$, and the bus takes $\mathbf{D_i}$ time to reach city i + 1.

At each city between 1 and **N** - 1, inclusive, you can decide to spend $\mathbf{T_s}$ time sightseeing before waiting for the next bus, or you can immediately wait for the next bus. You cannot go sightseeing multiple times in the same city. You may assume that boarding and leaving buses takes no time. You must arrive at city **N** by time $\mathbf{T_f}$ at the latest. (Note that you cannot go sightseeing in city **N**, even if you arrive early. There's nothing to see there!)

What is the maximum number of cities you can go sightseeing in?

Input

The input starts with one line containing one integer **T**, which is the number of test cases. **T** test cases follow.

Each test case begins with a line containing 3 integers, **N**, $\mathbf{T_s}$ and $\mathbf{T_f}$, representing the number of cities, the time taken for sightseeing in any city, and the latest time you can arrive in city **N**.

This is followed by **N** - 1 lines. On the i-th line, there are 3 integers, $\mathbf{S_i}$, $\mathbf{F_i}$ and $\mathbf{D_i}$, indicating the start time, frequency, and duration of buses travelling from city i to city i + 1.

Output

For each test case, output one line containing `Case #x: y`, where x is the test case number (starting from 1) and y is the maximum number of cities you can go sightseeing in such that you can still arrive at city **N** by time $\mathbf{T_f}$ at the latest. If it is impossible to arrive at city **N** by time $\mathbf{T_f}$, output `Case #x: IMPOSSIBLE`.

Limits

$1 \leq \mathbf{T} \leq 100$.

Small dataset

$2 \leq \mathbf{N} \leq 16$.
$1 \leq \mathbf{S_i} \leq 5000$.
$1 \leq \mathbf{F_i} \leq 5000$.
$1 \leq \mathbf{D_i} \leq 5000$.
$1 \leq \mathbf{T_s} \leq 5000$.
$1 \leq \mathbf{T_f} \leq 5000$.

Large dataset

$2 \leq \mathbf{N} \leq 2000$.
$1 \leq \mathbf{S_i} \leq 10^9$.
$1 \leq \mathbf{F_i} \leq 10^9$.
$1 \leq \mathbf{D_i} \leq 10^9$.
$1 \leq \mathbf{T_s} \leq 10^9$.
$1 \leq \mathbf{T_f} \leq 10^9$.

Sample

Input	Output
4 4 3 12 3 2 1 6 2 2 1 3 2 3 2 30 1 2 27 3 2 1 4 1 11 2 1 2 4 1 5 8 2 2 5 10 5000 14 27 31 27 11 44 30 8 20 2000 4000 3	Case #1: 2 Case #2: 0 Case #3: IMPOSSIBLE Case #4: 4

In the first test case, you can go sightseeing in city 1, catching the bus leaving at time 3 and arriving at time 4. You can go sightseeing in city 2, leaving on the bus at time 8. When you arrive in city 3 at time 10 you immediately board the next bus and arrive in city 4 just in time at time 12.

Problem B. Sherlock and The Matrix Game

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

Small input 13 points	Solve B-small
Large input 19 points	Solve B-large

Problem

Today, Sherlock and Watson attended a lecture in which they were introduced to matrices. Sherlock is one of those programmers who is not really interested in linear algebra, but he did come up with a problem involving matrices for Watson to solve.

Sherlock has given Watson two one-dimensional arrays A and B; both have length **N**. He has asked Watson to form a matrix with **N** rows and **N** columns, in which the j^{th} element in the i^{th} row is the product of the i-th element of A and the j-th element of B.

Let (x, y) denote the cell of the matrix in the x-th row (numbered starting from 0, starting from the top row) and the y-th column (numbered starting from 0, starting from the left column). Then a submatrix is defined by bottom-left and top-right cells (a, b) and (c, d) respectively, with a ≥ c and d ≥ b, and the submatrix consists of all cells (i, j) such that c ≤ i ≤ a and b ≤ j ≤ d. The sum of a submatrix is defined as sum of all of the cells of the submatrix.

To challenge Watson, Sherlock has given him an integer **K** and asked him to output the $\mathbf{K}^{th}$ largest sum among all submatrices in Watson's matrix, with **K** counting starting from 1 for the largest sum. (It is possible that different values of **K** may correspond to the same sum; that is, there may be multiple submatrices with the same sum.) Can you help Watson?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case consists of one line with nine integers **N**, **K**, $\mathbf{A_1}$, $\mathbf{B_1}$, **C**, **D**, $\mathbf{E_1}$, $\mathbf{E_2}$ and **F**. **N** is the length of arrays A and B; **K** is the ranking of the submatrix sum Watson has to output, $\mathbf{A_1}$ and $\mathbf{B_1}$ are the first elements of arrays A and B, respectively; and the other five values are parameters that you should use to generate the elements of the arrays, as follows:

First define $x_1 = \mathbf{A_1}$, $y_1 = \mathbf{B_1}$, $r_1 = 0$, $s_1 = 0$. Then, use the recurrences below to generate x_i and y_i for i = 2 to **N**:

- x_i = (**C***x_{i-1} + **D***y_{i-1} + $\mathbf{E_1}$) modulo **F**.
- y_i = (**D***x_{i-1} + **C***y_{i-1} + $\mathbf{E_2}$) modulo **F**.

Further, generate r_i and s_i for i = 2 to **N** using following recurrences:

- r_i = (**C***r_{i-1} + **D***s_{i-1} + $\mathbf{E_1}$) modulo 2.
- s_i = (**D***r_{i-1} + **C***s_{i-1} + $\mathbf{E_2}$) modulo 2.

We define $\mathbf{A_i} = (-1)^{r_i} * x_i$ and $\mathbf{B_i} = (-1)^{s_i} * y_i$, for all i = 2 to **N**.

Output

For each test case, output one line containing `Case #x: y`, where x is the test case number (starting from 1) and y is the $\mathbf{K}^{th}$ largest submatrix sum in the matrix defined in the statement.

Limits

1 ≤ **T** ≤ 20.
1 ≤ **K** ≤ min(10^5, total number of submatrices possible).
0 ≤ $\mathbf{A_1}$ ≤ 10^3.
0 ≤ $\mathbf{B_1}$ ≤ 10^3.
0 ≤ **C** ≤ 10^3.
0 ≤ **D** ≤ 10^3.
0 ≤ $\mathbf{E_1}$ ≤ 10^3.
0 ≤ $\mathbf{E_2}$ ≤ 10^3.
1 ≤ **F** ≤ 10^3.

Small dataset

1 ≤ **N** ≤ 200.

Large dataset

$1 \le \mathbf{N} \le 10^5$.

Sample

Input	Output
4 2 3 1 1 1 1 1 1 5 1 1 2 2 2 2 2 2 5 2 3 1 2 2 1 1 1 5 9 8 7 6 5 4 3 2 1	Case #1: 6 Case #2: 4 Case #3: 1 Case #4: 42

In case 1, using the generation method, the generated arrays A and B are [1, -3] and [1, -3], respectively. So, the matrix formed is
[1, -3]
[-3, 9]
All possible submatrix sums in decreasing order are [9, 6, 6, 4, 1, -2, -2, -3, -3].
As **K = 3**, answer is 6.

In case 2, using the generation method, the generated arrays A and B are [2] and [2], respectively. So, the matrix formed is
[4]
As **K = 1**, answer is 4.

In case 3, using the generation method, the generated arrays A and B are [1, 0] and [2, -1] respectively. So, the matrix formed is
[2, -1]
[0, 0]
All possible submatrix sums in decreasing order are [2, 2, 1, 1, 0, 0, 0, -1, -1].
As **K = 3**, answer is 1.

Problem C. Trash Throwing

Small input 17 points	Solve C-small
Large input 27 points	Solve C-large

Problem

Bob is an outstanding Googler. He loves efficiency, so he does everything well and quickly. Today, Bob has discovered that the trash can near his desk has disappeared! Sadly, this means that he has to use another nearby trash can instead. Since getting out of his seat to use the trash can would lower his productivity, Bob has decided to *throw* his trash into that trash can!

But there are many obstacles in the Google office. For example, it is rude if the thrown trash hits somebody, or the wall, or anything else. Bob hopes to throw the trash without touching any existing obstacles.

To simplify this problem, we will only consider the vertical plane that includes Bob and the trash can. Bob is at point (0, 0); the trash can is at point (**P**, 0). Moreover, there are **N** obstacles in the office; each of them is a single point, and the i-th one has coordinates ($\mathbf{X_i}$, $\mathbf{Y_i}$). The ceiling of the office is a line with the expression **y**=**H** in the plane. Since Bob is in one of the new high-tech floating offices, we do not consider the office floor in this problem; you do not need to worry about collisions with it. Bob will throw a piece of trash that is a circle with radius *R*. The center of the piece of trash starts off at (0, 0). When the piece of trash is thrown, the center of the piece of trash must follow the path of a parabola with the expression f(x)=*a*x(x-**P**), where 0 ≤ x ≤ **P**, and *a* can be any real number less than or equal to 0. The piece of trash is only considered thrown away when its center reaches the trash can's point, and it is not enough for some part of the piece of trash to just touch that point.

Bob is wondering: what is the largest piece of trash he can throw without hitting the ceiling or any obstacles? That is, we must find the maximum value of *R* for which there is at least one value *a* that satisfies the following: for any 0 ≤ x ≤ **P**, the Euclidean distance between (x, f(x)) and (x, H) is greater than *R*, and for each i, the Euclidean distance between the point (x, f(x)) and ($\mathbf{X_i}$, $\mathbf{Y_i}$) is greater than or equal to *R*.

Input

The input starts with one line containing one integer **T**, the number of test cases. **T** test cases follow. The first line of each test case contains three integers **N**, **P**, and **H**: the number of obstacles, the x-coordinate of the trash can, and the height of the ceiling. Then, there are **N** more lines; the i-th of those lines represents the i-th obstacle, and has two integers $\mathbf{X_i}$ and $\mathbf{Y_i}$, representing that obstacle's coordinates.

Output

For each test case, output one line `Case #x: y`, where x is the test case number (starting from 1) and y is a double representing the maximum radius **R**. Your answer will be considered correct if it is within an absolute or relative error of 10^{-4} of the correct answer. See the FAQ for an explanation of what that means, and what formats of real numbers we accept.

Limits

1 ≤ **T** ≤ 50.
2 ≤ **P** ≤ 1000.
2 ≤ **H** ≤ 1000.
0 < $\mathbf{X_i}$ < **P**.
0 < $\mathbf{Y_i}$ < **H**.

Small dataset

N = 1.

Large dataset

1 ≤ **N** ≤ 10.

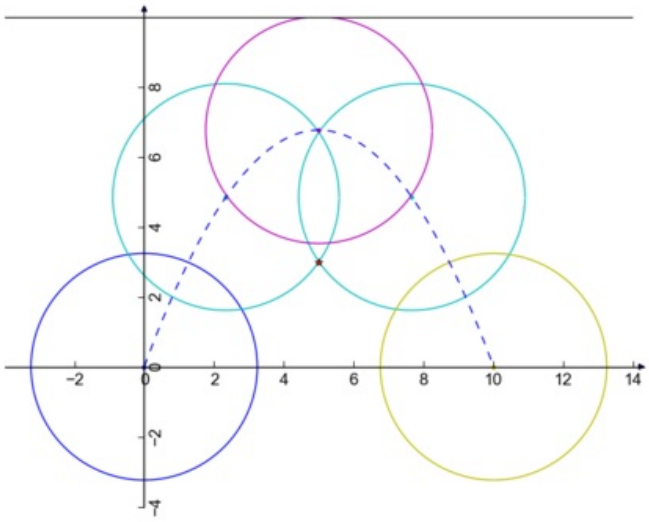
Sample

Input	Output

```
4           Case #1: 3.23874149472
1 10 10     Case #2: 4.00000000000
5 3         Case #3: 3.50000000000
1 10 10     Case #4: 2.23145912401
5 4
1 100 10
50 3
2 10 10
4 2
6 7
```

Note that the last sample case would not appear in the Small dataset.

The following picture illustrates Sample Case #1. Bob is at (0, 0), and the trash can is at (10, 0). There is a obstacle at point (5, 3), marked with a star. If Bob throws trash over the top of the obstacle, the maximal *R* is 3.2387, which requires an *a* of about -0.2705. If Bob throws trash under the obstacle, the maximal *R* is 3, which requires an *a* of 0. So the maximum *R* for this case is about 3.2387.



Sample Case #2 is like Sample Case #1, but the obstacle is one unit higher. Now, if Bob throws the trash under the obstacle, the maximal *R* is 4 (for $a = 0$). If he throws the trash over the obstacle, he can only use trash with a radius up to about 2.8306 (with $a = -0.4$). So the maximum *R* for this case is 4.