Problem A
เกมส์ผสมสิบ

(Time limit: 1 second)

เกมง่าย ๆ ที่สามารถใช้ฝึกเด็กในการหัดบวกตัวเลขและความไวคือเกมผสมสิบ เกมนี้จะเล่นบนตารางตัวเลข แบบสี่เหลี่ยมจัตุรัสที่มีตัวเลข 1-9 อยู่ในนั้น ผู้เล่นจะต้องหาว่ามีชุดตัวเลขมากกว่าหนึ่งตัวที่วางเรียงต่อกันในแนวตั้งหรือ แนวนอนที่บวกรวมกันได้สิบทั้งหมดกี่ชุดตัวอย่างเช่นจากตารางต่อไปนี้

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

จะนับชุดตัวเลขที่บวกกันได้สิบทั้งหมด 5 ชุด เป็นต้น

Input

อินพุตจะมีหลายชุด โดยในบรรทัดแรกจะเป็นจำนวนชุดของอินพุต ในแต่ละชุดของอินพุตบรรทัดแรกคือขนาดของตาราง เป็นเลขจำนวนเต็มมีค่าอยู่ระหว่าง 2 ถึง 100 และจะตามด้วยข้อมูลตารางนั้น ๆ โดยแสดงเป็นตัวเลขที่มีช่องว่างคั่นและการขึ้นบรรทัดใหม่ (ในอินพุทไฟล์ ตัวอย่างที่สองคือ รูปข้างบน)

Output

ในแต่ละชุดของอินพุต ให้แสดงเอาท์พุตเป็นเลขจำนวนเต็มที่แสดงถึงจำนวนชุดตัวเลขที่บวกกันได้สิบ

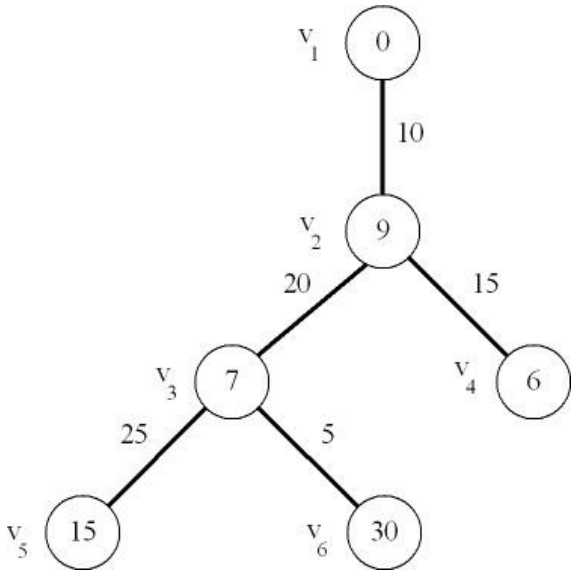
Example

Input	**Output**
3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 4 1 3 5 7 2 4 8 2 4 3 1 1 2 3 5 6 5 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2	4 5 10

Problem B
Burglary
(Time limit: 1 second)

A thief wants to steal as many valuable items in a tight security building as possible. In this tight security building n valuable items in the set $I = \{i_1, i_2, i_3,\ldots, i_n\}$ are kept in n rooms. Exactly one item is kept in any room. Rooms in this building are designed to have a tree structure $T = (V, E)$, where $V = \{v_1, v_2,\ldots, v_n\}$ is the set of vertices (rooms) and $E = \{e_1, e_2,\ldots, e_{n-1}\}$ is the set of edges (security measures). Each item has an associated value given by a function $P: I \rightarrow N$, where N is the set of natural numbers starting at 1. Each security measure has an associated cost of breaking in given by a function $C: E \rightarrow N$.

Note that, for a security reason, the item at vertex v_1 always has an item with value 0 and the edge $\{v_1, v_2\}$ always exists. In addition, vertex v_1 always has degree 1.



The picture illustrates a tree structure $T = (V, E)$, where $V = \{v_1, v_2,\ldots, v_6\}$ and $E = \{\{v_1, v_2\}, \{v_2, v_3\}, \{v_2, v_4\}, \{v_3, v_5\}, \{v_3, v_6\}\}$. Item at vertex v_2 has a value of 9. The security measure on edge $\{v_1, v_2\}$ has a penetration cost of 10, meaning that a thief has to pay 10 dollars to get the item in vertex v_2. An attack always starts from vertex v_1. A thief with an attacking budget B wants to find a simple path $<v_1, v_2,\ldots,v_i>$ such that the sum of the values of all items in this path is maximum and the sum of all penetration costs on this path must not exceed B. Your job is to write a program to help the thief find this path.

Input

The first line of the input file contains the number of test cases $t \leq 10$. Each test case has 4 lines. The first line contains the number of vertices $|V|$ and the given budget B. The second line contains the values for items $i_1, i_2, i_3,\ldots, i_n$ in this order. You can assume that item i_k is kept in vertex v_k. The third line contains $2(n-1)$ integers representing $e_1, e_2,\ldots, e_{n-1}$ in this order. The fourth line contains $n-1$ integers representing the penetration costs for edges $e_1, e_2,\ldots, e_{n-1}$, respectively. In all test cases $2 \leq n \leq 100{,}000$.

Output

The output file has *t* lines. The first line contains two integers representing the maximum sum of the values of all items and the remaining budget *B* for the test case 1. If budget *B* is exhausted, output 0. Similarly, the *t*-1 remaining lines report two integers for the subsequent test cases.

Example

Input	**Output**
2 4 6 0 5 30 15 1 2 2 3 2 4 2 8 4 6 60 0 9 7 6 15 30 1 2 2 3 2 4 3 5 3 6 10 20 15 25 5	20 0 46 14

Remark: The second test case in the input illustrates the example in the problem.

Problem C
Minion King Bob's Secret Book
(Time limit: 1 second)

When Minion King Bob wrote his secret book, he encrypted it. His method may not be sophisticated but it's enough to make his enemy understand nothing in his book. King Bob's method of encryption can be described in two phases in the following.

First Phase: Character Transposition
In the character transposition, King Bob's method first creates a table with k columns and $\lceil 26/k \rceil$ rows, for a given positive integer $k \leq 26$. Then, the characters *a-z* are written out in row-major order as illustrated in the following table.

	1	2	3	4	5	6
1	*a*	*b*	*c*	*d*	*e*	*f*
2	*g*	*h*	*i*	*j*	*k*	*l*
3	*m*	*n*	*o*	*p*	*q*	*r*
4	*s*	*t*	*u*	*v*	*w*	*x*
5	*y*	*z*				

Table 1: the table with $k = 6$.

Next, the character mapping is created by reading out characters in the table in reverse column major-order (last column first). For example, according to Table 1, character *a* is mapped with *f*, *b* with *l*, *c* with *r*, *d* with *x*, *e* with *e*, *f* with *k*, and so on. As a result, the mapping of the character transposition is illustrated as follows.

Character Position	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*
Original a-m	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*	*k*	*l*	*m*
Transposition	*f*	*l*	*r*	*x*	*e*	*k*	*q*	*w*	*d*	*j*	*p*	*y*	*c*

Character Position	13	14	15	16	17	18	19	20	21	22	23	24	25
Original n-z	*n*	*o*	*p*	*q*	*r*	*s*	*t*	*u*	*v*	*w*	*x*	*y*	*z*
Transposition	*i*	*o*	*u*	*b*	*h*	*n*	*t*	*z*	*a*	*g*	*m*	*s*	*y*

Table 2: the character transposition corresponding to Table 1 ($k = 6$).

Second Phase: Character Shifting.
In the character shifting, each of 26 characters after the transposition is shifted to the next k letters in alphabetical order. That is, after the transposition, character position *i* is changed to a new character at character position *i+k*. Note that the shifting for the last k characters can be done by having the first k characters in alphabetical order as the next k characters after

them. For example, for $k = 6$, character *a* is shifted to *g*, *b* to *h*, *c* to *i*, and so on. For the last 6 characters, character *u* is shifted to *a*, *v* to *b*, *w* to *c*, *x* to *d*, *y* to *e*, and *z* to *f*.

The complete character mapping after the character shifting is illustrated in the following table.

Original a-z	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*	*k*	*l*	*m*	…
Transposition	*f*	*l*	*r*	*x*	*e*	*k*	*q*	*w*	*d*	*j*	*p*	*y*	*c*	…
Shifting	*l*	*r*	*x*	*d*	*k*	*q*	*w*	*c*	*j*	*r*	*v*	*e*	*i*	…

Table 3: the complete character mapping corresponding to Table 2 (*k*=6).

Depending on k, King Bob used the method described above to encrypt texts in his secret book. For example, for $k = 6$, the text "ik zclov yuf" appearing in the book can be decrypted to the plain text "me thank you". You are asked by King Bob's enemy to write a program to decrypt this secret book.

Input

The first line of the input file contains the number of test cases $t \leq 28$, representing the number of lines in the book. This is followed by the input of each case one by one. Each case has two lines. The first line contains integer $k \leq 25$. The second line contains encrypted text which King Bob uses k in the method of encryption.

Output

The output has t lines. The first line contains the plain text decrypted from the first test case. Similarly, the $t - 1$ remaining lines contain the plain texts decrypted from the subsequent test cases.

Example

Input	**Output**
2 6 ik zclov yuf 6 ik mloz rlolol	me thank you me want banana

Problem D
A Remake of KPlumber

(Time limit: 3 seconds)

KPlumber was known as one of the most popular computer puzzle games in 1990s. The rule of the game is simple. It is played on a $n \times m$ rectangular (chessboard-like) grid of n columns and m rows, where every cell in the gird is assigned one of the five different tiles depicted in the following figure.

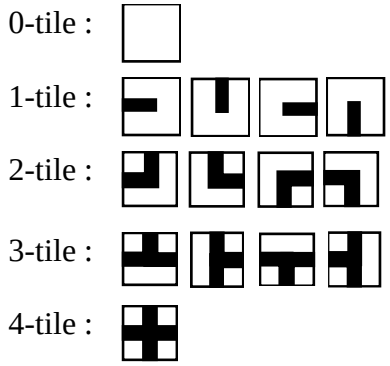
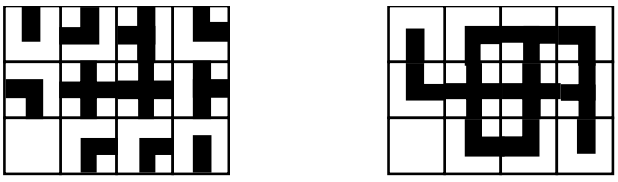


Fig. 1: Five possible types of tiles and their possible rotations

Each of these five possible types of tiles can be described as follows: a *0-tile* has no pipe; a *1-tile* contains a pipe that connects one side of the tile to its center; a *2-tile* contains pipes that connect two adjacent sides of the tile; a *3-tile* contains pipes that connects three adjacent sides of the tile; and a *4-tile* contains pipes that connects all sides of the tile. Each of these tiles can be rotated freely, but not moved from its place.

If there is a pipe running from the center of a tile to the middle of one of its sides, then this side is called *open*. Otherwise, this side is called *closed*. If the pipes are filled with water, then the system will possibly leak at an open side of some tile. The only way to prevent the leak is to have another tile with an open side in the adjacent cell, so that the water can flow on into the open pipe in the adjacent cell.



(a) (b)

Fig. 2: An initial configuration of the 4×3 grid (a) and its safe state (b).

In the initial configuration, all the tiles are rotated arbitrarily. If the player clicks on one of the tiles, then this tile makes a counter-clockwise rotation by 90 degrees. Note that four clicks of the same tile bring it back to the initial state. The goal of KPlumper is to bring the pipe system into a *safe state* by a number of clicking, where all pairs of tiles in adjacent cells either touch each other in open sides or touch each other in closed sides.

Now, Mr. Tim Gook, one huge fan of KPlumber, is trying to build a remake version of the game. Your job here as his assistant is to write a program to verify whether the game stage (the initial configuration of the game) designed by him has a safe state. Note that there is no way that one can win the game if a safe state does not exist.

Input

The first line of the input contains the number of test cases $t \leq 15$. This is followed by the input of each case one by one. For each case, the first line has two integers n and m, where $2 \leq nm \leq 2600$, indicating a number of columns and a number of rows in the grid, respectively. The next m lines, each of which contains n numbers from the set $\{0, 1, 2, 3, 4\}$, specify the initial configuration of tiles in the grid in the way that the cell at the i^{th} row and j^{th} column contains a k-tile if and only if the j^{th} number at the i^{th} line is k. Observe that the initial orientation of each *k*-tile is not known.

Output

The output has t lines. Each line contains YES or NO. If the initial configuration in the first test case has a safe state, then output YES. Otherwise, output NO. Similarly, the *t*-1 remaining lines report a safe state for the subsequent test cases.

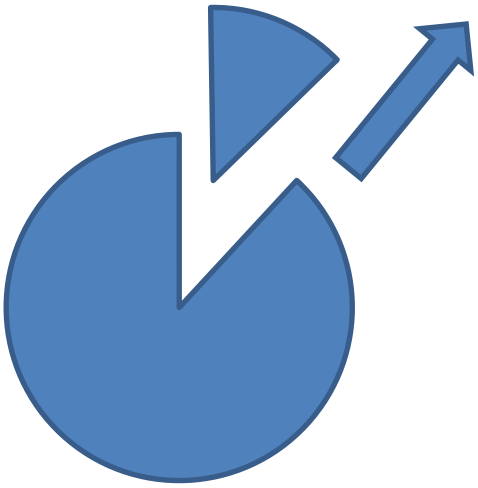
Example

Input	**Output**
2 2 2 0 1 1 3 4 3 1 2 3 2 2 4 4 3 0 2 2 1	NO YES

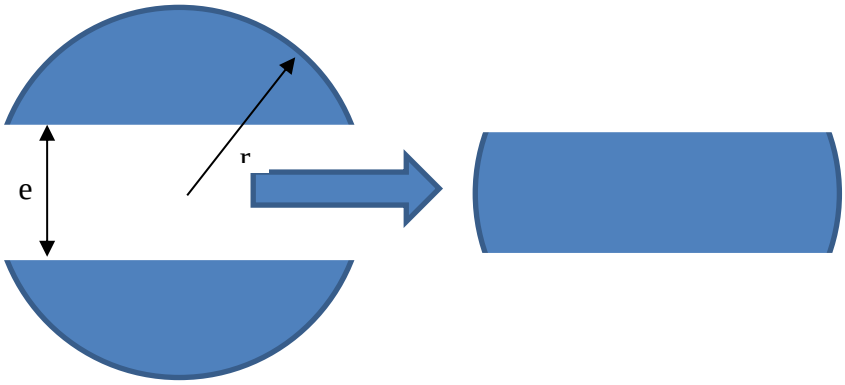
Remark: The second test case in the input illustrates the initial configuration in Fig 2.

Problem E
Let Them Eat Cake
(Time limit: 1 second)

In every birthday party, people will have to decide how to keep the leftover cake for the next day. Usually people will cut the cake just like a pie chart, i.e., as shown in picture below. Observe that this cake is so thin that it has no height.



The problem of this method is that the leftover cake will dry out and lose the deliciousness very fast. In 1906, a mathematician name Sir Francis Galton proposed[1] a new cake cutting method that can preserve the goodness of the cake.



In his cake-cutting method, we are going to cut the cake in the middle, just like the picture above. Then, the middle part will be eaten while the remaining two parts will be put together to avoid dry-out.

1 http://galton.org/essays/1900-1911/galton-1906-cake.pdf

E

Given the radius *r* of the cake and the width of the middle piece *e*, your job is to develop a program to find the ratio of the middle piece to the whole cake.

Input

The input consists of many test cases, the first line of the input will be the number of cases. Each case consists of two numbers, first number will be the radius (1 <= *r* <= 10,000) and the second number will be the middle piece width (1 <= *e* <= 5,000). Both *r* and *e* are integers.

Output

For each test case, the output ratio should be given as an integer number. In particular, the decimal part shall be removed. For example, if the ratio is 2.6, then the output should be 2.

Example

Input	**Output**
3 10 2 20 4 30 5	9 9 7

Problem F
Room

(Time limit: 1 second)

You are given an exhibition room with n booths spreading throughout the room. Room's style and arrangement of the booths are under the following criteria and assumptions. Interior angles of the room's corners must be less than 180 degrees each. At each corner of the room, there is always a booth setup since room's corners are golden locations for every merchant. Accordingly, the host of the exhibition requires a booth's location to be close to the room's center in order to set up an information desk of the exhibition. What this exhibition's host needs is that he needs you to help him find the booth(s) located in the center of the room. If there is more than one booth located in the room's center, report all such booths.

We define the distance function $d(p_1,p_2)$, where $p_1 = (x_1,y_1)$ and $p_2 = (x_2,y_2)$ are point coordinates.

$$d(p_1,p_2) = \sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$$

Let $d_{min}(b_c, b_1)$ be the minimum distance from the booth b_c to some booth b_1 at the room corner. Likewise, let $d_{max}(b_c, b_2)$ be the maximum distance from the booth b_c to some booth b_2 at the room corner. The booth at the center of the room is the booth b_c such that $d_{max}(b_c, b_2) - d_{min}(b_c, b_1)$ is minimized.

Input

There are $n + 1$ lines, $1 \leq n \leq 100$. The first line is a positive integer n indicating the number of booths in this exhibition. Each of the next n lines consists of two integers separated by one space character. The two integers in each line are a pair of coordinates (x, y) representing a booth's location.

Output

There are $m + 1$ lines. The first line is a positive integer m indicating the number of booths located in the middle of the exhibition room. Each of the next m lines consists of two integers separated by one space character. The two integers in each line are a pair of coordinates (x, y) representing location of a booth located at the center of the room.

Example

Input	**Output**
10 4 0 1 1 4 4 2 3 3 3 0 4 0 0 1 2 3 2 2 2	1 2 2

Problem G
How similar are they?

(Time limit: 1 second)

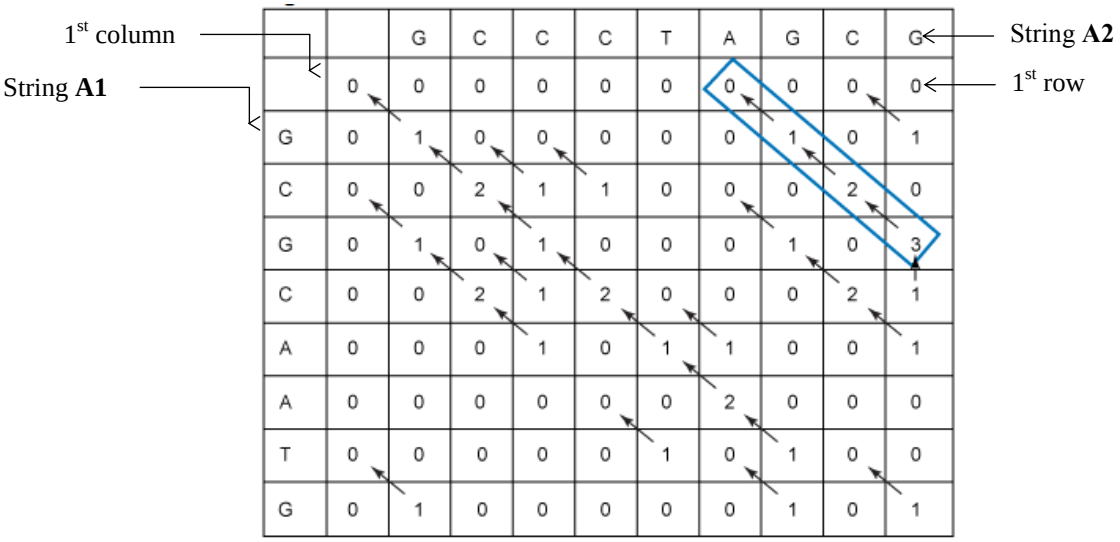
This problem is about strings and is probably one of the easiest problems.

One way of finding similarity between two strings of different sizes is to determine the most similar segments (or substring) of those two sequences. In order to quantify the similarity between two strings, a kind of the scoring system is used.

The overall similarity score of two strings depends on a cumulative comparison score of each character ***i*** in one string to each character ***j*** in another string. There are three possible outcomes.

- A "**Match**", when both characters are the same. A score is ***positive*** point. For example, we can award ***+1 point*** for each match.
- A "**Mismatch**", when the characters are NOT the same. A ***negative*** point will be used as a score. For example, we can penalize ***-1 point*** for each mismatch.
- A "**Gap**", when we consider inserting a space, "-", at either location ***i*** in one string or location ***j*** in another string. Another ***negative*** point will be used as a score. For example, penalize ***-2 point*** for each ***insertion*** whether in the first or the second string.

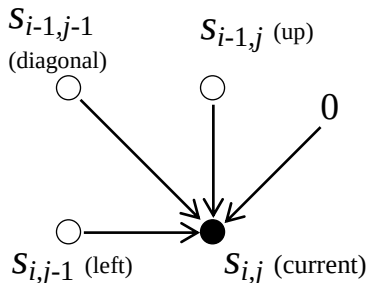
For example, if we want to find parts of string **A1**="**GCGCAATG**" and string **A2**="**GCCCTAGCG**" that are the most similar. We build a matrix to store similarity scores as shown below:



		G	C	C	C	T	A	G	C	G
	0	0	0	0	0	0	0	0	0	0
G	0	1	0	0	0	0	0	1	0	1
C	0	0	2	1	1	0	0	0	2	0
G	0	1	0	1	0	0	0	1	0	3
C	0	0	2	1	2	0	0	0	2	1
A	0	0	0	1	0	1	1	0	0	1
A	0	0	0	0	0	0	2	0	0	0
T	0	0	0	0	0	1	0	1	0	0
G	0	1	0	0	0	0	0	1	0	1

Scoring matrix

- We started by initializing all cells in the 1st row and the 1st column with 0
- Filling the score matrix. The score in each cell, $s_{i,j}$ where $i > 0$ and $j > 0$, can be calculated from three neighbors and the value **0** as follows:



$$s_{i,j} = max \begin{cases} 0 \\ s_{i-1,j} + \textbf{Gap score} \\ s_{i,j-1} + \textbf{Gap score} \\ s_{i-1,j-1} + \textbf{Match or Mismatch score} \end{cases}$$

- Arrows in the score matrix point back to which of those three neighbors we used to get the value for the current cell ***i,j***.
- After filling all cells, search for the cell with the **maximum score**. From that cell, starts tracing back with arrows until we found a cell with **0**.
    - If an arrow points back to $s_{i-1,j-1}$ then we output the character in row ***i*** from **A1** and character in column ***j*** from **A2** as results.
    - If the arrow points back to $s_{i-1,j}$ then we output the character in row ***i*** from **A1** and a space, **'-'**, from **A2** as results.
    - If the arrow points back to $s_{i,j-1}$ then we output a space, **'-'**, from **A1** and character in column ***j*** from **A2** as results.
- The output from backtracking process are two very similar (or possibly the same) segments from both input sequences.

From the example, we get "**GCG**" segment from both strings as results because **3** is the maximum score in the matrix and when tracing back there are three consecutive diagonal arrows until we found 0.

By shifting the string S1 to the right six characters, we can see that "**GCG**" segment from both strings is the answer.

S1 = GCGCAATG

S2 = GCCCTAGCG

Write a program to compare two strings with different sizes and return segments from both strings that give the best similarity score.

Input

The first line is the number of data sets, $n \leq 10$. The maximum size of all strings in the test cases is 100 characters.

- The following are *n* input data sets. Each data set contains
    - Line#1: integer scores for **Match**, **Mismatch** and **Gap** respectively, separated by space.
    - Line#2: string **A1**
    - Line#3: string **A2**
- Each input string can have different sizes and must have **NO** space in between.

Output

Each input data set gives a set of output that contains:

- Line#1: the best similarity score.
- Line#2: the output segment from A1.
- Line#3: the output segment from A2.
- ***Note***: it is possible to have *more than one pair of segments* that give the same similarity score.

Example

Input	**Output**
3 1 -1 -2 GCGCAATG GCCCTAGCG 1 -2 -2 actGGGctaacTGAct GGGaattaTGAtac 10 -5 -7 soften bestoftimes	3 GCG GCG 3 GGG GGG TGA TGA 33 s-oft stoft

Problem H
เกาะแห่งท้องฟ้า (Skypiea)

(Time limit: 1 second)

เกาะแห่งท้องฟ้าประกอบด้วยเกาะต่าง ๆ N เกาะ หมายเลขของเกาะคือ 0 ถึง N-1 ไวเปอร์ผู้นำเหล่านักรบแห่งแชนโดราต้องการส่งนักรบซึ่งมีจำนวนไม่จำกัดที่อยู่ที่เกาะหมายเลข 0 ไปจัดการก๊อดเอเนลูที่เกาะหมายเลข N-1 โดยในการเดินทาง นักรบจะต้องเดินทางข้ามเกาะเหล่านี้ การเดินทางจากเกาะ i ไป j สามารถเดินทางได้หากมีสะพานเชื่อมระหว่างเกาะ ทั้งนี้สะพานจากเกาะ i ไป j จะใช้เดินทางได้จากเกาะ i ไป j เท่านั้น จะไม่สามารถใช้เดินทางจากเกาะ j ไป i ได้ นอกจากนี้ยังพบว่าสะพานที่ใช้บนเกาะแห่งท้องฟ้าล้วนเก่าแล้ว แต่ละสะพานจึงรองรับจำนวนคนที่ผ่านได้จำกัดเมื่อคนผ่านครบจำนวนแล้ว สะพานจะพังและใช้ไม่ได้อีกต่อไป

ก๊อดเอเนลูรู้ว่าจะมีผู้บุกรุกจึงส่งฟ้าฝ่ามาทำร้ายเหล่านักรบขณะข้ามสะพาน หากนักรบต้องการข้ามสะพานอย่างปลอดภัย จะต้องเป่านักหวีดเรียกใช้บริการจอมเทพกันโฟลเพื่อช่วยคุ้มครองขณะข้ามสะพาน จอมเทพจะเก็บค่าข้ามสะพานต่อคนซึ่งแต่ละสะพานเก็บไม่เหมือนกันแล้วแต่ความยากง่าย ไวเปอร์ผู้นำทางการรบของชนเผ่าต้องการที่จะส่งกองกำลังไปยังเกาะที่ก๊อดเอเนลูอยู่ให้มากที่สุดและเสียเงินให้กับจอมเทพกันโฟลสำหรับปกป้องกองกำลังให้น้อยที่สุด หน้าที่ของคุณคือออกแบบโปรแกรมเพื่อคำนวณว่าไวเปอร์สามารถส่งกองกำลังไปได้มากที่สุดกี่คน และถ้ามีวิธีในการส่งกองกำลังไปได้มากที่สุดมากกว่า 1 วิธี ให้เลือกวิธีที่เสียเงินสำหรับส่งคนกลุ่มนี้ให้จอมเทพน้อยที่สุด

Input

บรรทัดแรกเป็นจำนวนชุดทดสอบ $T \leq 10$ แต่ละชุดบรรทัดที่หนึ่งเป็นจำนวนเมือง $N \leq 100$ และจำนวนสะพาน $M \leq 99{,}000$ อีก M บรรทัดแต่ละบรรทัดประกอบด้วย i j k l แทนสะพานเชื่อมจากเกาะ i ไปเกาะ j รองรับได้ k คนและค่าใช้จ่ายในการเรียกจอมเทพ l หน่วยต่อคน

Output

มีสองบรรทัดต่อชุดทดสอบ บรรทัดที่ 1 แทนจำนวนคนมากที่สุดที่สามารถไปได้ บรรทัดที่ 2 แทนค่าใช้จ่ายสำหรับคนในบรรทัดแรกที่น้อยที่สุด

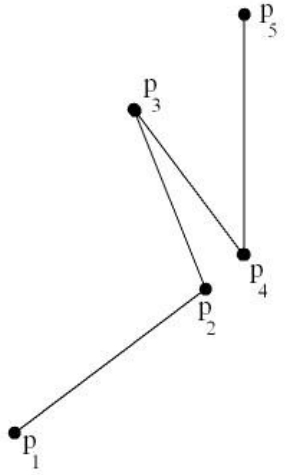
Example

Input	**Output**
2 3 2 0 1 5 1 1 2 4 1 5 5 0 1 2 1 0 2 2 2 1 3 2 1 2 3 2 3 3 4 3 1	4 8 3 12

Problem I
Turning Points

(Time limit: 1 second)

In geometry a line segment $\overline{p_1 p_2}$ can be defined by its two end points p_1 and p_2. In this problem we are only interested in points on a plane. That is, a point is defined by its corresponding coordinate (x, y). You are given a sequence of n points $P = <p_1, p_2, p_3, \ldots, p_n>$. This sequence represents a series of continuous line segments $\overline{p_1 p_2}\ \overline{p_2 p_3}\ \overline{p_3 p_4}\ \overline{p_4 p_5} \cdots \overline{p_{n-1} p_n}$ and you are asked to compute the directions left or right of each turn of the n segments.



For instance, you are given a sequence of 5 points $P = <p_1, p_2, p_3, p_4, p_5>$. This sequence represents a series of continuous line segments $\overline{p_1 p_2}\ \overline{p_2 p_3}\ \overline{p_3 p_4}\ \overline{p_4 p_5}$ and you are asked to compute the directions (left or right) of each respective turn of the n-1 segments.

Input

Let $3 \leq |P| \leq 200{,}000$. The first line of the input file contains an integer representing the number of test cases $m \leq 10$. The second line contains $2n_1$ integers representing n_1 coordinates for the first test case. The third line of the input file contains $2n_2$ integers representing n_2 coordinates for the second test case and so on for the subsequent test cases.

Output

The first line contains a string made up of characters L or R of length n_1-2. This string represents the directions (**L**eft or **R**ight) of the n_1-1 line segments in the first test case. That is, the first character identifies the direction of the first 3 points p_1, p_2, p_3 and the second character identifies the direction of the second 3 points p_2, p_3, p_4 and so on. Similarly, the second line contains a string made up of characters L or R of length n_2-2 and so on for the subsequent test cases.

Example

Input	**Output**
2 5 1 6 2 7 1 1 1 3 3 2 6 4 4 4 7	R LRL

Remark: The second test case illustrates the example in the problem.