The 37th Annual

ACM International Collegiate Programming Contest ASIA Regional - Daejeon

Problem Set

Please check that you have 12 problems and 21 sheets (excluding additional materials).

A.	Accelerator	(2 pages)
B.	Contour Maps	(2 pages)
C.	Critical 3-Path	(2 pages)
D.	Dot Number	(1 page)
E.	Palindrome	(1 page)
F.	Pandora	(2 pages)
G.	Pattern Lock	(2 pages)
H.	Pole Arrangement	(1 page)
I.	Rock Paper Scissors	(2 pages)
J.	Slicing Tree	(3 pages)
K.	Sports Repoters	(1 pages)
L.	Square Annulus	(2 pages)

The 37th Annual

ACM International Collegiate Programming Contest
Asia Regional - Daejeon

Problem A
Accelerator

A huge particle accelerator will be constructed in Daedeok Innopolis. The accelerator may be considered as a huge electric circuit, containing a lot of bridges. The bridges must be connected by wires. Particularly, there are red and blue bridges, and each red bridge must be connected by a wire to one of the blue bridges. The goal is to minimize the total length of wires connecting red bridges with blue ones.

More precisely, we have a circle representing the accelerator, and there are red and blue points on the circle representing the red bridges and the blue bridges. The number of red points is less than or equal to the number of blue points and all red and blue points are distinct. Each red point must be connected to a blue point with a wire. Two or more red points cannot be connected to the same blue point. The length of wire is the arc length between the connected points. So the goal is to minimize the sum of the arc lengths between matched points.

For convenience, we use a circle of perimeter n, and consider only n locations on the circle such that the arc length between two adjacent locations is 1. The locations are numbered by 0, 1, …, $n-1$ clockwise. Each red or blue point is at one of these locations, that is, the location of a red or blue point is represented as one of the integer numbers 0, 1, …, $n-1$. The length between arbitrary pair of red and blue points is always an integer. For example, Figure 1 shows 3 red points and 3 blue points lying on a circle with 12 locations. In an optimal matching of this example, the red points at location 1, 3, and 9 are matched with the blue points at location 5, 4, and 10, respectively. So the minimum sum of arc lengths between the matched points is 6.

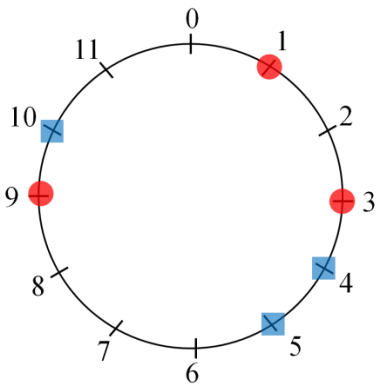


Figure 1.

Given red and blue points on a circle sorted clockwise, write a program to match the red points and the blue ones minimizing the sum of the arc lengths between matched points.

Input

Your program is to read from standard input. The input consists of T test cases. The number of test cases T is given in the first line of the input. Each test case starts with a line containing three integers, n, a and b ($1 \le n \le 1{,}000{,}000$, $1 \le a \le b \le 1{,}000{,}000$, $2 \le a + b \le n$), where n is the number of locations considered on the circle, a is the number of red points and b is the number of blue points. In the second line of each test

case, a sorted integers $x_1, x_2, \ldots, x_a$ are given, representing the locations of red points. In the third line of each test case, b sorted integers $y_1, y_2, \ldots, y_b$ are given, representing the locations of blue points. Then $0 \le x_i, y_j \le n-1$, $i = 1, \ldots, a$ and $j = 1, \ldots, b$, and all x_i's and y_j's are distinct. There is a single space between two integers in same line.

Output

Your program is to write to standard output. Print exactly one line for each test case. The line should contain the minimum of the sum of arc lengths between the matched red and blue points.

The following shows sample input and output for two test cases.

Sample Input	Output for the Sample Input
2 12 3 3 1 3 9 4 5 10 12 3 4 1 4 7 3 6 8 11	6 4

The 37th Annual

ACM International Collegiate Programming Contest
Asia Regional - Daejeon

Problem B
Contour Maps

Hallyeohaesang national park is located in the south of the Korean Peninsula. *Hallyeohaesang* is a unique marine ecosystem which extends along 120km. The park is studded with more than 360 islands, large and small. Specially, *Hallyeosudo* which is known as the most beautiful waterway ever has 69 uninhabited islands and 30 inhabited islands spread out across the sea like jewels.

Mr. Kim, a park superintendent, plans to conduct a survey about the elevations of all islands of the park. He has contour maps of the islands. A contour map shows elevations above sea level and surface features of the land by means of contour lines. A contour line is a closed curve which joins points of equal elevation above a given level (see Figure 1).

The contour lines of the islands in the park, do not intersect each other, and are simple convex closed curves like ellipses. In order to use a computer, he first converted the original contour maps to corresponding digital contour maps. In a digital contour map, a contour line is represented as the boundary of a rectilinear convex polygon (see Figure. 2). A rectilinear polygon is a polygon whose edges are either horizontal or vertical. A rectilinear polygon P is called convex if the intersection of the interior of P and a vertical or horizontal line is either empty or a line segment. A level is a predefined measure for elevation. The outermost contour line is at level 1. If the closest contour line surrounding a contour line c is at level k, then c is at level $k+1$. In Figure 2, the level of the highest contour line that we can see in the map is 5.

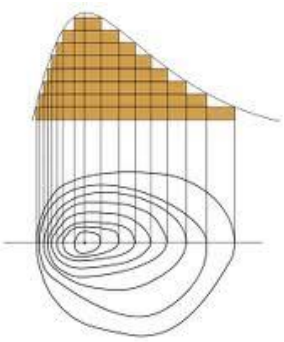


Figure1. Contour lines

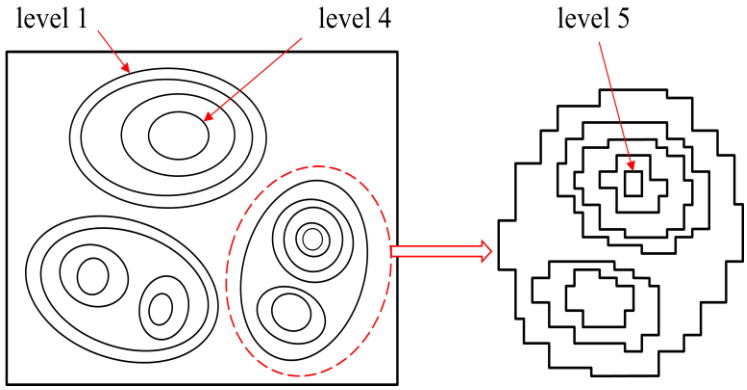


Figure 2. A contour map

Given a contour map with n contour lines that are represented as the boundaries of rectilinear convex polygons, write a program that computes the level of the highest contour line in the map.

Input

Your program is to read from standard input. The input consists of T test cases. The number of test cases T is given in the first line of the input. Each test case starts with a line containing an integer m ($1 \le m \le 20{,}000$), where m is the number of rectilinear convex polygons representing contour lines. In the next m lines of each test case, each line contains $2k+1$ integers $k, x_1, y_1, x_2, y_2, \ldots, x_k$, and y_k ($4 \le k \le 100$, $1 \le x_i, y_i \le 10^9$),

where k is the number of vertices of a rectilinear convex polygon Q and (x_i, y_i)'s are coordinates of k vertices of Q, which are in counterclockwise order. Notice that the boundaries of any two rectilinear convex polygons do not intersect each others.

Output

Your program is to write to standard output. Print exactly one line for each test case. The line should contain an integer, the level of the highest contour line in the map.

The following shows sample input and output for two test cases (see Figure 3).

Sample Input	Output for the Sample Input
2 5 4 5 3 14 3 14 12 5 12 4 6 4 13 4 13 11 6 11 4 7 5 12 5 12 10 7 10 4 8 6 11 6 11 9 8 9 4 9 7 10 7 10 8 9 8 4 4 10 7 11 7 11 8 10 8 6 3 7 5 7 5 9 6 9 6 12 3 12 6 8 5 9 5 9 6 12 6 12 9 8 9 10 9 2 9 3 14 3 14 11 7 11 7 8 6 8 6 4 4 4 4 2	5 3

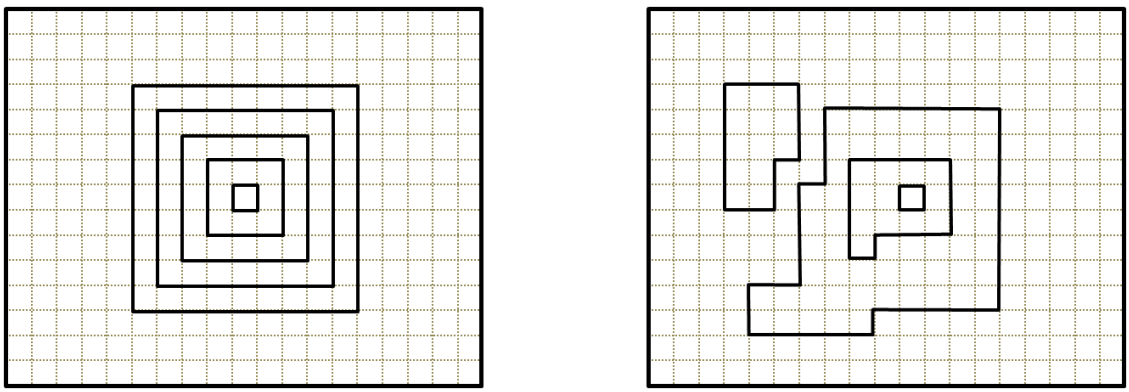


Figure 3. Illustrations for Sample Input.

The 37th Annual

ACM International Collegiate Programming Contest
Asia Regional - Daejeon

Problem C
Critical 3-Path

The PERT (Project Evaluation and Review Technique) chart, a graphical tool used in the field of project management, is designed to analyze and represent the set of tasks involved in completing a given project. Edges in PERT chart represent tasks to be performed, and edge weights represent the length of time required to perform the task. For vertices u, v, w of a PERT chart, if edge (u,v) enters vertex v and edge (v,w) leaves v, then task (u,v) must be performed prior to task (v,w). A path through a PERT chart represents a sequence of tasks that must be performed in a particular order. Note that there is no cycle in the PERT chart. A critical path is a longest path in PERT chart, corresponding to the longest time to perform an ordered sequence of tasks. The weight of a critical path is a lower bound on the total time to perform all the tasks in a project.

A *3-path* of six distinct vertices s_1, s_2, s_3, t_1, t_2, t_3 in a PERT chart is defined as follows:

(1) A 3-path consists of three paths P_i from vertex s_i to vertex t_i for i =1, 2, 3.

(2) The paths P_1, P_2, P_3 are vertex-disjoint, i.e., no two of the paths have vertices in common.

The length of a 3-path is the sum of the length of the 3 paths P_1, P_2, P_3. A *critical 3-path* of six distinct vertices in a PERT chart is a 3-path of maximum length over all 3-paths.

For example, a critical 3-path $\{P_1, P_2, P_3\}$ of a graph in Figure 1, where P_1 is a path from vertex 3 to vertex 15, P_2 is a path from vertex 4 to vertex 16, and P_3 is path from vertex 5 to vertex 17, is as follows:

$$P_1 : 3 \rightarrow 6 \rightarrow 11 \rightarrow 15$$
$$P_2 : 4 \rightarrow 7 \rightarrow 9 \rightarrow 12 \rightarrow 16$$
$$P_3 : 5 \rightarrow 8 \rightarrow 13 \rightarrow 17$$

The length of the critical 3-path is 128.

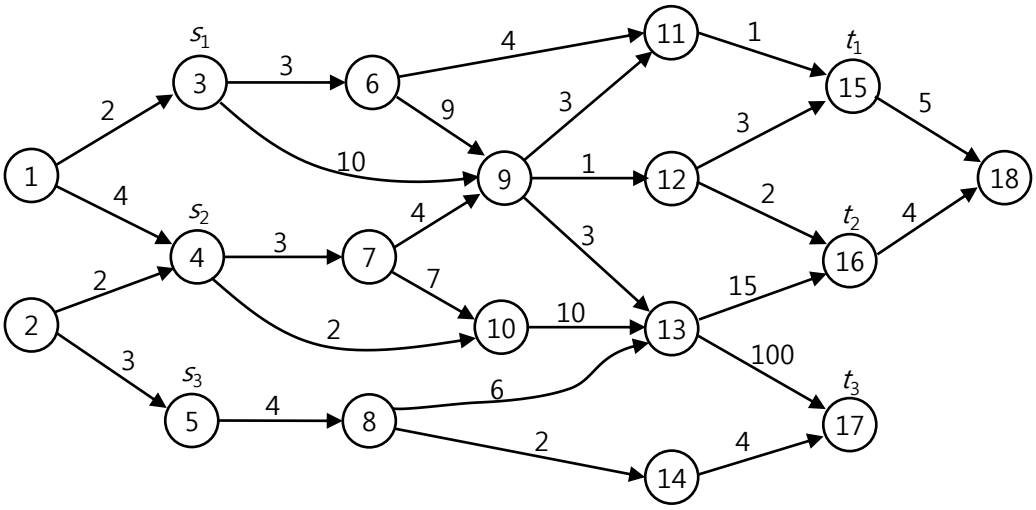


Figure 1. A sample PERT chart

Given a graph corresponding to a PERT chart and six distinct vertices, write a program to find the length of critical 3-path of the graph corresponding to the six vertices.

Input

Your program is to read from standard input. The input consists of T test cases. The number of test cases T is given in the first line of the input. Each test case starts with a line containing two integers, n and m ($6 \le n \le 100$, $n-1 \le m \le n(n-1)/2$), where n is the number of vertices and m is the number of edges. Every input node is numbered from 1 to n. Next line contains six integers s_1, s_2, s_3, t_1, t_2, t_3, where all six integers are distinct. In the following m lines, the weight of the directed edges are given; each line contains three integers, u, v, and W ($1 \le W \le 100{,}000$), where W is the weight of an edge from vertex u to v. You may assume that $u < v$.

Output

Your program is to write to standard output. Print exactly one line for each test case. The line should contain the length of critical 3-path $\{P_1, P_2, P_3\}$, where P_i is a path from s_i to t_i ($1 \le i \le 3$). If there does not exist a critical 3-path, print 0.

The following shows sample input and output for two test cases.

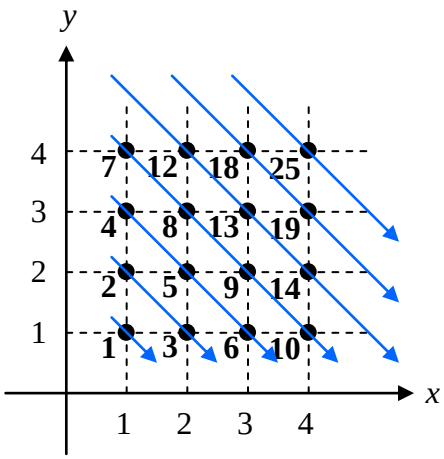
Sample Input	Output for the Sample Input
2 18 27 3 4 5 15 16 17 1 3 2 1 4 4 2 4 2 2 5 3 3 6 3 3 9 10 4 7 3 4 10 2 5 8 4 6 11 4 6 9 9 7 9 4 7 10 7 8 13 6 8 14 2 9 11 3 9 12 1 9 13 3 10 13 10 11 15 1 12 15 3 12 16 2 13 16 15 13 17 100 14 17 4 15 18 5 16 18 4 6 5 1 2 3 4 5 6 1 2 1 2 3 1 3 4 1 4 5 1 5 6 1	128 0

The 37th Annual

ACM International Collegiate Programming Contest
Asia Regional - Daejeon

Problem D
Dot Number

You are given an infinite number of dots in the first quadrant of two-dimensional Cartesian plane. The x and y coordinates of each dot are positive integers. You are going to count the dots by assigning a natural number to every dot in the diagonal order as in the figure.



The unique number assigned to each dot is called the *dot number*. The dot number of (x, y) is denoted as $\#(x, y)$. For example, the figure shows the dot numbers for some dots, $\#(1, 1) = 1$, $\#(2, 1) = 3$, $\#(2, 2) = 5$, and $\#(4, 4) = 25$, etc.

The addition of dots is defined by those of corresponding coordinates, i.e. $(a, b) + (c, d) = (a + c, b + d)$. Since the dot number determines the actual coordinates of the dot, the addition of two dots can also be denoted by that of two dot numbers. For example, the addition of the dots $(1, 1)$ and $(2, 2)$ can be denoted by $1 + 5$ since $\#(1, 1) = 1$, $\#(2, 2) = 5$, and the addition result is 13 since $\#(3, 3) = 13$.

You are to write a program which adds two dots represented by their dot numbers. Your program should decipher the dot numbers, add the dots, and print the dot number of the resulting dot.

Input

Your program is to read from standard input. The input consists of ***T*** test cases. The number of test cases ***T*** is given in the first line of the input. Each test case consists in one line containing two dot numbers separated by a space. The dot numbers are greater than 0 and less than 10,000.

Output

Your program is to write to standard output. Print exactly one line for each test case. The line should contain an integer, the dot number of the result of the dot addition. Beware that the resulting dot number may greater than or equal to 10,000.

The following shows sample input and output for two test cases.

Sample Input	Output for the Sample Input
2 1 5 3 9	13 26

The 37th Annual
ACM International Collegiate Programming Contest
Asia Regional - Daejeon

Problem E
Palindrome

A palindrome is a word which can be read the same way in either direction. For example, the following words are palindromes: `civic`, `radar`, `rotor`, and `madam`.

You have found a note that contains *k* words. Later you found out that the note contains a clue to the password for accessing the most secret server where ICPC problems are stored. Unfortunately, the password is not written as it is. It is the concatenation of two different strings out of the *k* words, and it is a palindrome. For example, you can find the palindrome `abababa` from the five words `aaba`, `ba`, `ababa`, `bbaa`, and `baaba` by concatenating `ababa` and `ba`.

Write a program to find the password from *k* given words.

Input
Your program is to read from standard input. The input consists of *T* test cases. The number of test cases *T* is given in the first line of the input. Each test case starts with a line containing one integer $\boldsymbol{k}$ ($\mathbf{1} \leq \boldsymbol{k} \leq \mathbf{100}$), where $\boldsymbol{k}$ is the number of words in the note. In the next *k* lines of each test case, each line contains a word in ASCII characters between `a` and `z`. The sum of lengths of strings for each case is equal to or smaller than 10,000.

Output
Your program is to write to standard output. For each test case, if there is no palindrome obtained by concatenating two strings, print a single line containing `0`. Otherwise, print the palindrome. If there are two or more palindromes, just print one of them.

The following shows sample input and output for two test cases.

Sample Input	Output for the Sample Input
2 5 aaba ba ababa bbaa baaba 3 abc bcd cde	abababa 0

The 37th Annual
ACM International Collegiate Programming Contest
Asia Regional - Daejeon

Problem F
Pandora

An unmanned robot, named Pandora, launched by KARA (Korea Astronomy Research Association) arrived at Mars. Pandora found an unidentified architecture whose boundary is a simple rectilinear polygon.

A rectilinear polygon is a polygon whose edges are either horizontal or vertical. That is, at each vertex of the polygon, the interior angle formed by its two incident edges is either 90° or 270°, as shown in Figure 1. We say that a rectilinear polygon is *simple* if (1) each vertex is incident to exactly two edges and (2) there are no edges that intersect each other except at their end vertices.

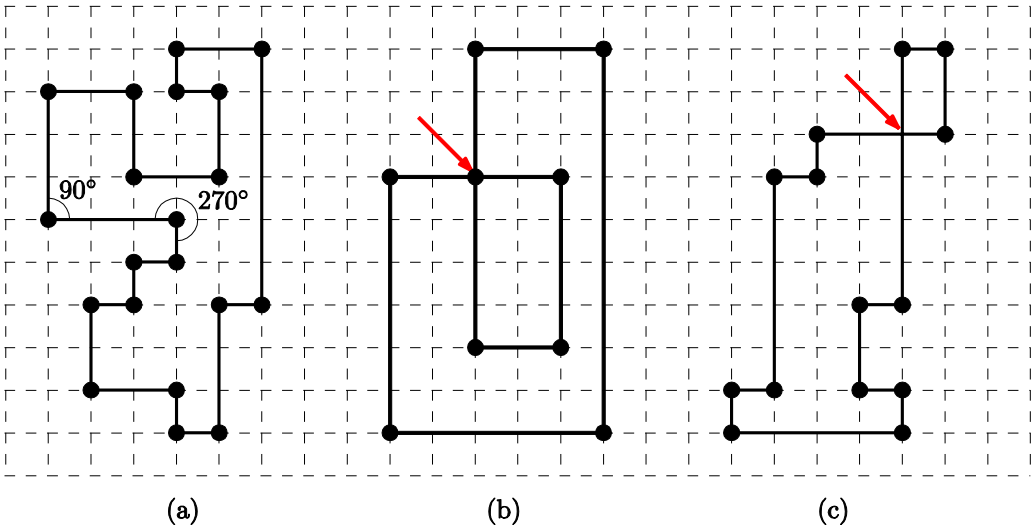


Figure 1. (a) A simple rectilinear polygon. (b)-(c) Non-simple rectilinear polygons because (b) there is a vertex having four incident edges, or (c) there is a pair of edges crossing each other.

To figure out the complete shape of the boundary of the simple rectilinear polygon, Pandora moved along the boundary in counterclockwise direction until it came back to the starting position. Also it sent to KARA geometric values on the boundary such as the length of the edges and turning angles (90° or 270°) at the vertices. But, due to the instability of the wireless communication between Pandora in Mars and KARA in Earth, only the angle information arrived at KARA. Thus KARA must guess the geometric shape of the boundary of the simple rectilinear polygon from the angle information only.

More formally, the angle information is represented as an *angle sequence* S of **`L`** and **`R`**. The letter **`L`** means a *left turn* (with respect to the moving direction) at a vertex, i.e., the corresponding vertex has an angle 90° and the letter **`R`** means a *right turn* at a vertex of angle 270°. For example, the angle sequence **`LLLL`** represents an axis-parallel rectangle. Figure 2 shows another example.

Let l be the number of **`L`**'s in S, and let r be the number of **`R`**'s in S. Since S is obtained by moving along the boundary of the simple rectilinear polygon in counterclockwise direction, it is true that $l = r + 4$, where $l \geq 4$ and $r \geq 0$. Conversely, if we have a sequence S with $l = r + 4$, then we can always draw one or more simple rectilinear polygons having $(l + r)$ vertices from S whose angle sequences in counterclockwise direction are

exactly *S*. The final task of KARA is to know the *monotonicity* of the simple rectilinear polygons drawn from *S*.

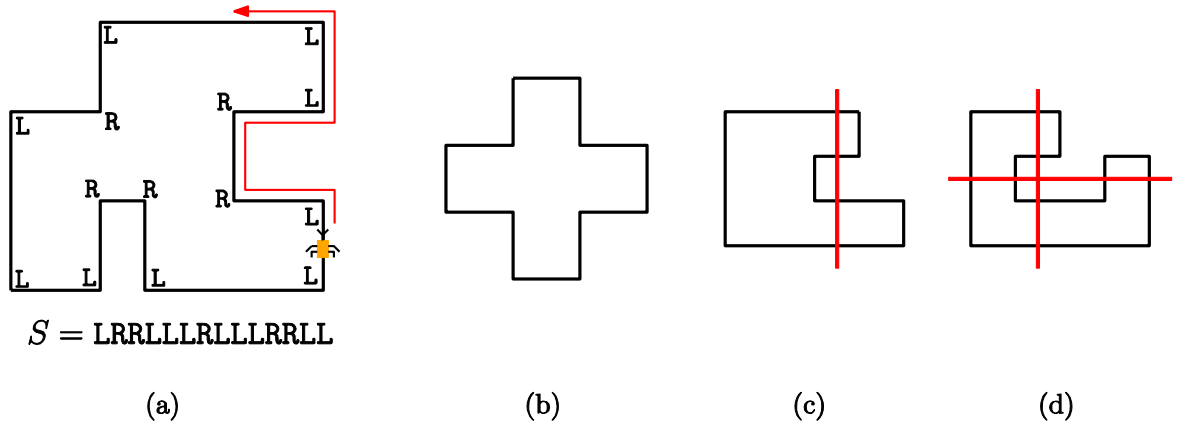


Figure 2. (a) A simple rectilinear polygon and its angle sequence ***S***. (b)-(d) Examples: (b) Monotone to two axes. (c) Monotone to one axis. (d) Not monotone to any axis.

A simple rectilinear polygon is *monotone* with respect to X-axis (resp., Y-axis) if any line orthogonal to X-axis (resp., Y-axis) intersects the polygon in (at most) one connected segment. As illustrated in Figure 2, simple rectilinear polygons as in Figure 2(b)-(c) are monotone to one or two axes, but some polygon as in Figure 2(d) is not monotone to any axis. KARA already proved that all the simple rectilinear polygons drawn from *S* have the same monotonicity property. You should output how many axes the simple rectilinear polygons drawn from *S* are monotone to. The output will be one of 0, 1, or 2, so you should output 0 if they are not monotone to any axis, 1 if they are monotone to exactly one axis, and 2 if they are monotone to two axes.

Input

Your program is to read from standard input. The input consists of *T* test cases. The number of test cases *T* is given in the first line of the input. Each test case contains a string ***S*** consisting of **`L`** and **`R`** such that the number of **`L`**'s is four more than the number of **`R`**'s in ***S***, and the length of ***S*** is inclusively between **4** and 100,000.

Output

Your program is to write to standard output. For each test case, print a line containing the number of axes, one of 0, 1, or 2, to which the simple rectilinear polygons drawn from *S* are monotone.

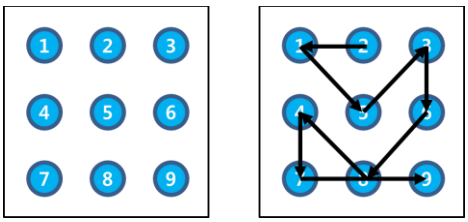
The following shows sample input and output for three test cases.

Sample Input	Output for the Sample Input
3 LRLLLRLL LLLLLLRR LRRLLLRLLLRRLL	2 1 0

The 37th Annual
ACM International Collegiate Programming Contest
Asia Regional - Daejeon

Problem G
Pattern Lock

Pattern lock security is generally used in Android handsets instead of a password. The pattern lock can be set by joining points on a 3×3 matrix in a chosen order. The points of the matrix are registered in a numbered order starting with 1 in the upper left corner and ending with 9 in the bottom right corner. This pattern must involve at least 4 points to set, cannot be disconnected and each point number can be used at most once. So the pattern of the lock screen in Figure 1(b) would be 2-1-5-3-6-8-4-7-9.

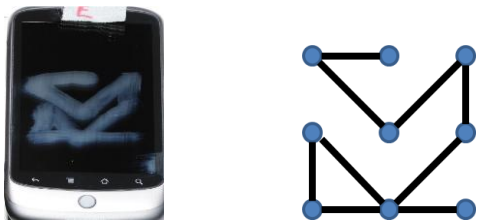


(a) (b)

Figure 1. (a) Android pattern lock screen with overlaid identification numbers on contact points. (b) A valid pattern lock.

In Figure 1(b), since the point 8 is already visited, you can connect from point 7 to point 9 directly. If the point 8 is not visited yet, you cannot connect from point 7 to point 9 directly.

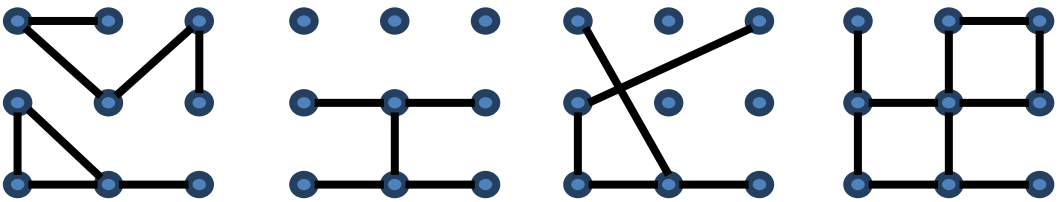
Though Chulsoo has completely forgotten his pattern, he can get his pattern image as a geometric graph from his smudged smartphone screen such as Figure 2.



(a) (b)

Figure 2. (a) A pattern image. (b) A geometric graph of the pattern image.

For example, let's consider four geometric graphs in Figure 3. Since the graph in Figure 3(a) is disconnected, this pattern is not possible to construct. Even though the graph in Figure 3(b) is connected, this pattern cannot be constructed. That is, there is no sequence joining points that makes the given pattern. The pattern given in Figure 3(c) can be constructed by the point-joining sequence 1-8-9-7-4-3 only. The pattern given in Figure 3(d) can be constructed by 4-5-6-3-2-8-9-7-1 or 8-5-2-3-6-4-1-7-9.



(a) (b) (c) (d)

Figure 3. The geometric graphs obtained from smudged smartphone screen.

You are going to find Chulsoo's pattern lock from his pattern image. Given a pattern image as a geometric graph, write a program to find a possible pattern lock sequence.

Input

Your program is to read from standard input. The input consists of $\boldsymbol{T}$ test cases. The number of test cases $\boldsymbol{T}$ is given in the first line of the input. Each test case starts with a line containing an integer $\boldsymbol{e}$ ($\mathbf{3} \leq \boldsymbol{e} \leq \mathbf{24}$), where $\boldsymbol{e}$ is the number of edges of the geometric graph. In the next $\boldsymbol{e}$ lines of each test case, the $\boldsymbol{i}$-th line contains two integers $\mathbf{s}_i, \boldsymbol{d}_i$ ($\boldsymbol{i} = \mathbf{1}, \mathbf{2}, \ldots, \boldsymbol{e}$ **and** $\mathbf{1} \leq \mathbf{s}_i, \boldsymbol{d}_i \leq \mathbf{9}$), which represent an edge between $\mathbf{s}_i$ **and** $\boldsymbol{d}_i$.

Output

Your program is to write to standard output. For each test case, if it can be possible to recover a pattern code, print a point joining sequence which makes a pattern. Otherwise, print "`IMPOSSIBLE`".

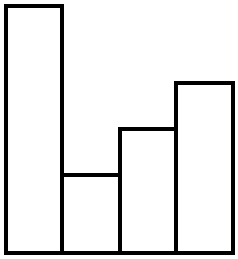
The following shows sample input and output for six test cases.

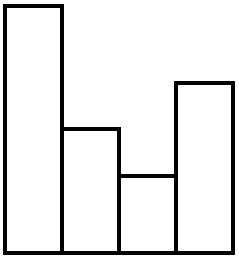
Sample Input	Output for the Sample Input
6 3 5 8 7 8 8 9 3 5 8 7 8 8 9 6 4 5 5 6 4 7 5 8 7 8 8 9 6 1 2 2 3 2 5 4 5 5 6 3 6 5 1 2 2 3 2 5 4 5 5 6 5 1 4 4 7 8 9 6 9 6 5	5 8 9 7 5 8 7 9 5 8 9 7 4 6 2 5 4 6 3 1 IMPOSSIBLE IMPOSSIBLE

The 37th Annual
ACM International Collegiate Programming Contest
Asia Regional - Daejeon

Problem H
Pole Arrangement

There are n poles of height 1, 2, ... , n in a row. If you look at these poles from the left side or the right side, smaller poles are hidden by taller poles. For example, consider the two arrangements of 4 poles in the next figure:





For each arrangement, only one pole can be seen from the left, and two poles can be seen from the right.

You are to write a program to calculate the number of arrangements of n poles such that seen from the left you see l poles and seen from the right you see r poles.

Input
Your program is to read from standard input. The input consists of T test cases. The number of test cases T is given in the first line of the input. Each test case consists of a line containing three integers, n, l, and r ($1 \leq l, r \leq n \leq 20$), where n is the number of poles and l(resp. r) is the number of poles that can be seen from the left(resp. right).

Output
Your program is to write to standard output. Print exactly one line for each test case. The line should contain the number of arrangements of poles for the test case.

The following shows sample input and output for four test cases.

Sample Input	Output for the Sample Input
4 4 1 2 4 1 1 5 2 4 20 2 1	2 0 4 6402373705728000

The 37th Annual

ACM International Collegiate Programming Contest
Asia Regional - Daejeon

Problem I
Rock Paper Scissors

We have robots who can play the "Rock Paper Scissors (RPS)" game. This game consists of ***k*** successive rounds. Each robot keeps a predefined sequence of *k* hands in a string of length *k*. In the game the disqualified robots will be eliminated and the game will go on with the remaining robots. The game is over if only one winner remains. If there are more than one robots survived after *k* rounds, the game should be declared a "draw".

Let R, P and S denote Rock, Paper and Scissor, respectively. If the hand string is "RSPSRSSP", then the robot will show "Rock" in the first round and "Scissors" in the next round. And in the final 8th round the robot will show "Paper" only if he is not disqualified during the game. In the game if only one winner remains after the *i*-th round competition, then the robot RPS game terminates in the *i*-th round and you should report the winner robot. The game may not terminate depending on the RPS sequences after *k* rounds have passed. Then you should report 0(zero) to declare a "draw" ending.

Input

Your program is to read from standard input. The input consists of *T* test cases. The number of test cases *T* is given in the first line of the input. Each test case starts with the integer *N* to denote the number of robots participating. Then the predefined RPS strings of length *k* are given in the following N lines one by one. Note that $2 \leq N \leq 10$ and *k*, the length of the RPS sequence string, is restricted to $3 \leq \boldsymbol{k} \leq 30$. For each test case, the robots are numbered 1 to *N* sequentially from top.

Output

Your program is to write to standard output. Print exactly one integer to denote the robot number of the one winner, if it exists. If there is no winner after the *k* rounds competition, then you should print zero(0) to denote the "draw" ending.

The following shows sample input and output for three test cases.

Sample Input	Output for the Sample Input
3 5 RPSSSPR SSRPRPS PRSSRSP RRRPSPP SSSSSRP 4 RPSPSPSPRPRPSR SPSSRRRSSRPRRR RSPRPPPPSSRPSR PRRSSSRRPRSRRR 3 SPPPSS SPRRRR SSSSPP	2 0 3

The 37th Annual

ACM International Collegiate Programming Contest
Asia Regional - Daejeon

Problem J
Slicing Tree

VLSI circuits are too complex to design without CAD tools since the complexity of many VLSI circuits is in the order of millions of transistors today. In order to reduce the complexity of the design process for a VLSI circuit, the whole process is broken into several intermediate phases. Among them is a physical design phase. In the physical design phase, the basic components of the circuit are usually thought of as rectangular modules. The physical design phase itself consists of several steps. One of the steps, which is most crucial for the performance of the circuit, is a floorplan/placement step. Actual floorplan/placement problem in the VLSI circuit design is very complex. However, here, you are asked to deal with a simple version of the problem, in which each component of a VLSI circuit can be viewed as a rectangle in the plane.

A simple version of the floorplan/placement problem is to place n rectangles in the plane in axis-parallel orientation such that some given constraints are satisfied. Information about relative locations among rectangles is given as constraints. Relative location means that rectangle R_i $(1 \leq i \leq n)$, should be placed either below (or above) rectangle R_j $(1 \leq j \leq n)$, or to the left (or right) hand side of rectangle R_j. For each rectangle R_i, the coordinates of its lower left corner and its upper right corner are represented as (x_i^{ll}, y_i^{ll}) and (x_i^{ur}, y_i^{ur}), respectively. That rectangle R_i is located below rectangle R_j means $y_i^{ur} \leq y_j^{ll}$. Similarly, that rectangle R_i is to the left of rectangle R_j means $x_i^{ur} \leq x_j^{ll}$. Notice that each rectangle can be placed rotated by 90°. Figure 1 shows two example cases in which rectangle R_1 is below rectangle R_2.

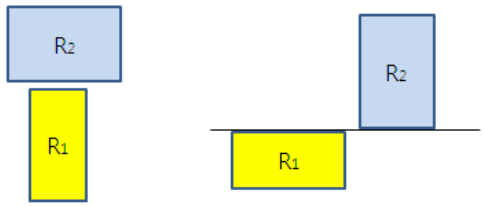


Figure 1. Two example cases to illustrate $\boldsymbol{R_1}$ is below $\boldsymbol{R_2}$

The constraints regarding relative locations among the rectangles are represented as a binary tree called 'slicing tree.' A slicing tree represents how the plane is partitioned and which rectangle should be placed in each sub-region. Each internal node of the slicing tree is labeled as either Ⓗ or Ⓥ. Each external node of the slicing tree is labeled with a rectangle identification number i $(1 \leq i \leq n)$. The label Ⓗ or Ⓥ for an internal node means that the sub-region on the plane is partitioned either by a horizontal line or by a vertical line. For example, if the slicing tree is as shown in Figure 2, it means that the plane is partitioned by a horizontal line and that rectangle R_1 should be placed below rectangle R_2. Notice that both of the placements shown in Figure 1 meet the constraint shown in Figure 2.

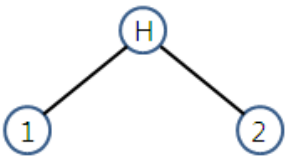


Figure 2. An example of slicing tree for 2 rectangles

On the other hand, if the slicing tree is as shown in Figure 3, it means that the plane is partitioned by a vertical line and that rectangle R_1 should be placed to the left hand of rectangle R_2 .

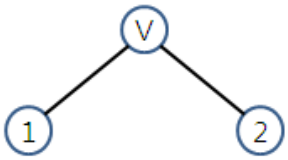


Figure 3. Another example of slicing tree for 2 rectangles

If any internal node N_k in the slicing tree is labeled as Ⓗ, all the rectangles belonging to the left sub-tree rooted at N_k should be placed below any rectangles belonging to the right sub-tree rooted at N_k. Similarly, if any internal node N_k in the slicing tree is labeled as Ⓥ, all the rectangles belonging to the left sub-tree rooted at N_k should be placed to the left of any rectangle belonging to the right sub-tree rooted at N_k. For example, Figure 4 shows a slicing tree and two different corresponding placements.

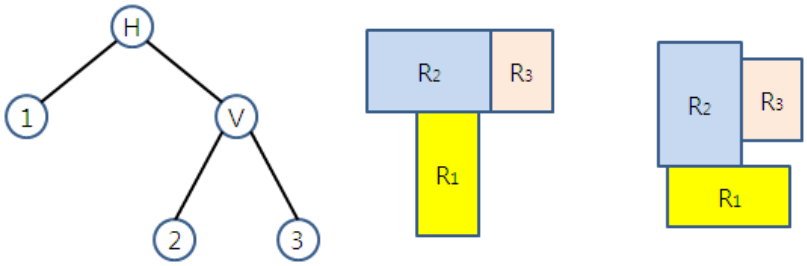


Figure 4. A slicing tree and its two different corresponding placements

Let 🅁 denote the minimum rectangle which encloses all the n rectangles when a placement is determined. Such enclosing rectangles corresponding to the placements shown in Figure 4 are shown in Figure 5 with dotted lines.

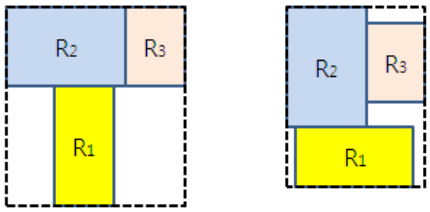


Figure 5. Minimum enclosing rectangles

Given information regarding a slicing tree and the rectangles' dimensions, your program should determine the location for each rectangle such that the placement meets the given constraints and such that the area of the enclosing rectangle 🅁 is as small as possible. Notice that the area of the enclosing rectangle 🅁 can be affected by the orientation of each rectangle.

Input

Your program is to read from standard input. The input consists of T test cases. The number of test cases T is given in the first line of the input. Each test case starts with a line containing an integer $\boldsymbol{n}$ ($\boldsymbol{1 \le n \le 1,000}$), where $\boldsymbol{n}$ is the number of rectangles. In the following $\boldsymbol{n}$ lines, dimensions of $\boldsymbol{n}$ rectangles are given, each line for each rectangle. The $\boldsymbol{i}$-$\boldsymbol{th}$ ($\boldsymbol{1 \le i \le n}$) line contains two integers, $\boldsymbol{w}$ and $\boldsymbol{h}$ ($\boldsymbol{1 \le w, h \le 500}$), $\boldsymbol{w}$ for width and $\boldsymbol{h}$ for height of rectangle $\boldsymbol{i}$. In the next line the information regarding a slicing tree is given. The

information is represented as a list consisting of $2n-1$ items separated by spaces, which is obtained by traversing the slicing tree in post-order. Each item of the list is either a label for an internal node or for an external node. The label for an internal node is either H or V. The label for an external node is an integer i ($1 \le i \le n$), which is the rectangle identification number.

Output

Your program is to write to standard output. Print exactly one line for each test case. For each test case, find a placement such that the given constraints regarding relative locations among rectangles are satisfied and the area of the enclosing rectangle ℝ is as small as possible. Then print the area of the rectangle ℝ for each test case. You can assume that the resulting area of ℝ is less than 10^9 for each test case.

The following shows sample input and output for two test cases.

Sample Input	Output for the Sample Input
2 5 1 5 4 2 3 3 1 3 5 4 2 1 V 3 5 H 4 V H 6 4 2 5 7 7 2 4 4 1 4 5 3 2 3 H 1 5 V 6 4 H V V	65 105

The 37th Annual
ACM International Collegiate Programming Contest
Asia Regional - Daejeon

Problem K
Sports Reporters

A news agency wishes to cover n sporting events that will take place soon. For each event E_i, $1 \le i \le n$, its starting time s_i, duration time d_i, and geographical site g_i are known in advance. In addition, the travel time $t_{i,j}$ from g_i to g_j is also known for all $1 \le i,j \le n$, where $t_{i,j} = t_{j,i}$ and $t_{i,j} \le t_{i,k} + t_{k,j}$ for every $1 \le i,j,k \le n$. In gathering news from an event, the news agency wants one reporter to fully take responsibility for the entire period of the event. This means that two events E_i and E_j may be assigned to the same reporter if $s_i + d_i + t_{i,j} \le s_j$ or $s_j + d_j + t_{j,i} \le s_i$.

Under these constraints, a manager of the agency tries to identify a maximum subset of events such that no pair of events in the subset can be assigned to the same reporter. Write a program that solves the manager's problem.

Input
Your program is to read from standard input. The input consists of T test cases. The number of test cases T is given in the first line of the input. Each test case is described by a number of lines. The first line of a test case contains an integer n, indicating the number of sporting events ($1 \le n \le 1{,}000$). Then, the second line contains n integers s_1, s_2, ..., s_n separated by spaces, where s_i is the starting time of event E_i ($1 \le s_i \le 1{,}000{,}000$). Similary, the third line contains n integers d_1, d_2, ..., d_n, where d_i is the duration time of E_i ($1 \le d_i \le 1{,}000{,}000$). In the following n lines, the i-th line contains $n - i + 1$ integers $t_{i,i}$, $t_{i,i+1}$, ..., $t_{i,n}$, where $t_{i,j}$ is the travel time from g_i to g_j ($0 \le t_{i,j} \le 1{,}000{,}000$). You may assume $t_{i,i} = 0$ for every i.

Output
Your program is to write to standard output. Print exactly two lines per each test case. The first line should contain the maximum number k of events in which no two can be assigned to the same reporter. Then, the second line should contain the indices of such k events (i.e., i for E_i), separated by a space. If there are multiple solutions, just output any one of them.

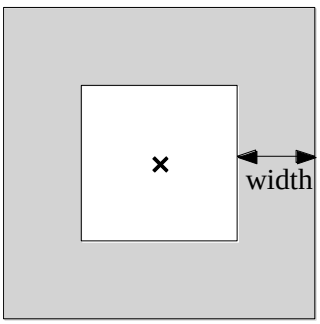
The following shows sample input and output for two test cases.

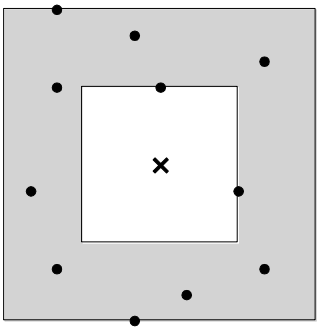
Sample Input	Output for the Sample Input
2 3 7 8 9 1 1 1 0 2 2 0 1 0 2 7 12 3 2 0 2 0	3 2 3 1 1 2

The 37th Annual
ACM International Collegiate Programming Contest
Asia Regional - Daejeon

Problem L
Square Annulus

A *square annulus* is the planar shape contained between two concentric axis-parallel squares, i.e., two squares with a common center whose sides are parallel to the *x*- and *y*-axes. More precisely, it means the area inside the bigger square but outside the smaller one, including their boundaries. The *width* of a square annulus is defined to be half the difference of the side lengths of its two squares. See the left figure below, in which a square annulus is depicted as gray region with the center marked by ×.





You are given *N* points in the plane and you have to find a square annulus **A** of minimum width that contains all the *N* given points. The right figure above shows an example of a square annulus of minimum width containing all given points (marked by dots).

Your program is to compute the width of a square annulus **A** that contains all the *N* given points. You can exploit the following fact, which has been shown by a German research group:

There exists a square annulus **A** *of minimum width containing N given points*
such that its bigger square is a smallest axis-parallel square containing the N points.

Because the bigger square *S* of **A** must contain all the *N* points, the above fact means that *S* can be assumed to have the minimum side length among all axis-parallel squares containing the *N* points; in other words, if *L* is the side length of *S*, then there is no axis-parallel square containing the *N* points with side length smaller than *L*.

Remark that there can be many such squares of the same side length with *S* containing the given points. Also, note that the width of **A** can be zero when the two squares defining **A** are the same; or, the smaller square of **A** may have side length zero, so the width of **A** can be as large as half the side length of its bigger square.

Input

Your program is to read from standard input. The input consists of T test cases. The number T of test cases is given in the first line of the input. From the second line, each test case is given in order, consisting of the following: a test case contains an integer $N(1 \leq N \leq 100{,}000)$, the number of input points, in its first line, and is followed by N lines each of which consists of two integers inclusively between $-1{,}000{,}000$ and $1{,}000{,}000$, representing the x- and y-coordinates of a point in the plane. Two consecutive integers in one line are separated by a single space and there is no empty line between two consecutive test cases.

Output

Your program is to write to standard output. Print exactly one line for each test case. The line should contain a single value, representing the minimum width of an axis-parallel square annulus **A** containing the N input points. The value to be printed should consist of the whole integer part of the computed width, a decimal point, and exactly one digit after the decimal point. So, if necessary, you should round off the computed width to one decimal place.

The following shows sample input and output for two test cases.

Sample Input	Output for the Sample Input
2 11 -2 -2 1 -4 3 -3 6 -2 5 1 6 6 2 5 -3 1 -2 5 -2 8 1 7 12 -11 0 15 -4 10 -13 5 -12 -4 19 0 17 8 30 -10 -23 -9 -17 6 27 -8 23 -11 -8	3.0 13.5