The 40th Annual

ACM International Collegiate Programming Contest ASIA Regional - Daejeon

Problem Set

Please check that you have 12 problems and 20 sheets (excluding additional materials).

The 40th Annual

ACM International Collegiate Programming Contest Asia Regional - Daejeon

Problem A

Coin Swap

Time Limit: 0.5 Seconds

A graph $G = (V, E)$ has a set V of vertices and a set E of edges, and it is connected, that is, there is a path between every pair of vertices in G. Each vertex in G has a color which is either black or white. A coin is placed on each vertex in G and each coin also has a black or white color. A pair of adjacent vertices can swap their current coins, which is called a *swap operation*.

A swap operation is carried out in such a way that one edge in G is selected and the end vertices of the edge exchange their coins. Through these swap operations, black coins and white ones should be placed on black vertices and white ones, respectively, which is the goal of the problem. It is assumed that the number of black (and white) coins lying on the vertices is equal to the number of the black (and white, resp.,) vertices. That is, we can always achieve the goal of the problem.

Given a graph G, colors of the vertices of G, and colors of coins initially placed on the vertices, your program is to find the minimum number of the swap operations achieving the goal.

For example, in Figure 1, the colors of vertices 1 and 2 are black and the others are white. The coins placed on the vertices are drawn as rectangles with their colors. Then after three swap operations as shown in Figure 1, the colors of coins and vertices agree at each vertex.

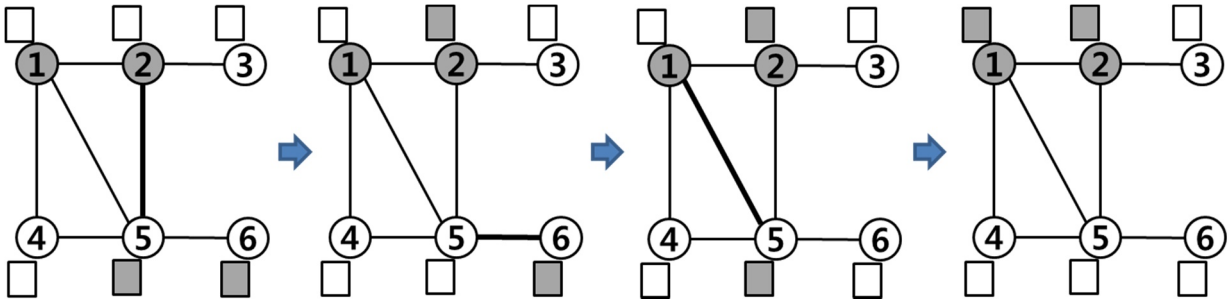


Figure 1. An example of swap operations.

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 ($1 \leq n \leq 500$, $n - 1 \leq m \leq \frac{n(n-1)}{2}$), where n and m represent the number of vertices and edges, respectively, of a connected graph G. The vertices are numbered from 1 to n. In each of the following m lines, there are two integers x and y ($1 \leq x < y \leq n$), which means there is an edge (x, y) connecting the vertices x and y in G. In the next line, there are n integers c_i ($c_i = 0$ or 1, $i = 1, \ldots, n$), separated by a single space, where c_i is the color of the vertex i. In the next line, there are n integers d_i ($d_i = 0$ or 1, $i = 1, \ldots, n$), separated by a single space, where d_i is the color of a coin initially placed on the vertex i. If $c_i = 0$ or $d_i = 0$, it means black, and if $c_i = 1$ or $d_i = 1$, it means white.

Output

Your program is to write to standard output. Print exactly one line for each test case. The line should contain an integer representing the minimum number of the swap operations such that every vertex shall get a coin of the same color.

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

Sample Input	Output for the Sample Input
3 4 4 1 2 1 3 2 4 3 4 0 1 1 0 0 1 0 1 6 7 1 2 1 5 1 4 2 3 2 5 4 5 5 6 1 1 0 0 0 0 0 0 0 0 1 1 6 5 1 2 2 3 3 4 4 5 5 6 1 0 1 1 0 0 0 0 1 1 0 1	1 3 5

The 40th Annual

ACM International Collegiate Programming Contest
Asia Regional - Daejeon

Problem B
Equilibrium State
Time Limit: 1 Second

When the sum of forces exerted on an object, say P, is 0 as shown in Figure 1, the object will not move. It is said that the object is in an equilibrium state.

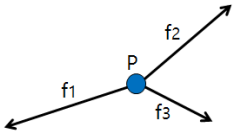


Figure 1. Forces exerted on object P

Suppose a situation as shown in Figure 2. There are two fixed points F_1 and F_2 on the x-axis. Object P is connected by two springs S_1 and S_2. Spring S_1 connects F_1 and P and spring S_2 connects F_2 and P. Let's denote w_1 and w_2 to be the elastic coefficients of springs S_1 and S_2, respectively, which denote how stiff the springs are. Let's also denote $x(F_1)$ and $x(F_2)$ to be the x-coordinates of F_1 and F_2, respectively. Then the force exerted on P by S_1 is $w_1 \times (x(P) - x(F_1))$ by Hooke's rule, where $x(P)$ is the x-coordinate of P. Similarly, the force exerted on P by S_2 is $w_2 \times (x(P) - x(F_2))$.

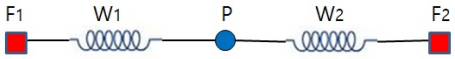


Figure 2. An example to illustrate an equilibrium state

For example, in a situation as shown in Figure 2, if $x(F_1) = 0$, $x(F_2) = 7$, $w_1 = 3$, $w_2 = 4$, and P is in an equilibrium state, then we can determine the location of P, i.e., $x(P) = 4$.

Consider a similar situation where several objects are connected by multiple springs in a two-dimensional plane as shown in Figure 3. Assume there are k fixed points $F_1, \ldots, F_k$ and n objects $P_1, \ldots, P_n$. Also, assume there are m springs $S_1, \ldots, S_m$ with elastic coefficients $w_1, \ldots, w_m$, respectively. Each of the springs is used to connect either a fixed point and an object or two objects. If every object is connected to at least two springs, then all the objects will finally get into an equilibrium state. Once all the objects get into an equilibrium state, we can determine the location of every object in the plane.

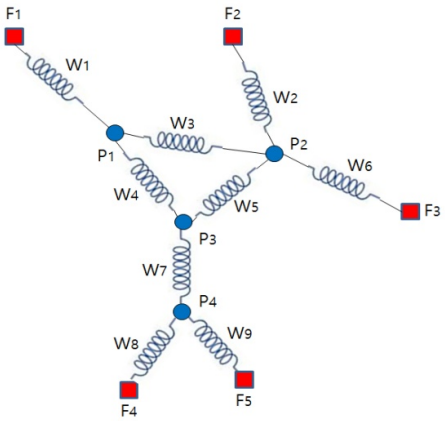


Figure 3. Another example to illustrate an equilibrium state in the plane

You are asked to make a program to determine the locations of n objects in an equilibrium state using given information: the locations of k fixed points, the elastic coefficients of m springs, and the interconnection information between objects and fixed points.

You can assume:

1. All objects are connected to at least two springs and any pair of objects and fixed points are connected through one or more springs.
2. There is at most one spring between either a pair of (object, object) or a pair of (object, fixed point).
3. There are at least three non-collinear fixed points.
4. When all objects are in an equilibrium state, no two springs will cross each other and no two objects will locate at the same position.
5. No two fixed points have the same coordinates.

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, k ($3 \leq k \leq 100$), m ($3 \leq m \leq 3{,}000$), and n ($1 \leq n \leq 1{,}000$), where k is the number of fixed points, m is the number of springs, n is the number of objects. In the following k lines, each line contains the integer coordinate (x_i, y_i) $(-10{,}000 \leq x_i, y_i \leq 10{,}000)$ of fixed point $F_i (1 \leq i \leq k)$. In the following m lines, each line contains three integers w_i, u_i, and v_i $(1 \leq i \leq m)$, where $w_i (1 \leq w_i \leq 100)$ is the elastic coefficient of spring S_i, and u_i and v_i are the indices of either a fixed point or objects which are connected to spring S_i. If u_i is negative then it means that fixed point F_{-u_i} $(1 \leq -u_i \leq k)$ is connected to spring S_i. On the other hand, if u_i is positive then it means that object P_{u_i} $(1 \leq u_i \leq n)$ is connected to spring S_i. Similarity holds for v_i.

Output

Your program is to write to standard output. For each test case, print first "`Test case number : `" followed by the test case number as shown in the following sample. Then print number i $(1 \leq i \leq n)$ starting from 1 to n followed by the coordinate (x_i, y_i) of object P_i in each of the following n lines. You print the coordinates of object P_i with two digits after decimal point after rounding off from the third digit. Three numbers (i, x_i, y_i) in each line should be separated by a blank. If each value of the coordinate is within an error range, 0.01, it will be considered correct.

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

Sample Input	Output for the Sample Input
3 4 5 2 -2 7 -6 2 2 -5 6 -1 4 -1 1 3 1 -2 1 1 2 2 2 -4 5 2 -3 5 9 4 -81 67 -4 67 59 0 -40 -68 0 -65 2 -1 1 5 -2 2 3 1 2 1 1 3 7 3 2 4 2 -3 8 4 3 4 -4 4 5 4 -5 4 26 12 -50 50 -50 -50 50 -50 50 50 11 -1 10	Test case number : 1 1 -2.95 3.89 2 2.38 -2.89 Test case number : 2 1 -25.05 30.11 2 5.56 18.79 3 -5.02 -9.75 4 -11.77 -39.71 Test case number : 3 1 -0.60 5.21 2 -1.74 -1.85 3 7.33 -2.39 4 -11.65 -2.98 5 18.28 3.35 6 -3.43 -7.02 7 12.44 6.52 8 7.37 -7.65 9 20.92 -13.70 10 -19.60 16.71 11 27.77 20.55 12 -22.86 -21.43

14 -4 11 12 -2 12 12 -3 9 13 10 1 14 1 7 11 7 11 11 4 2 13 2 3 10 3 5 11 12 6 15 6 8 12 8 9 11 10 4 13 1 2 11 7 3 15 11 5 10 4 12 14 2 6 12 3 8 14 5 9 10 1 3 12 7 5 11 4 6 11 3 6 11 3 9	

The 40th Annual
ACM International Collegiate Programming Contest
Asia Regional - Daejeon

Problem C
Grid
Time Limit: 1 Second

You are given a grid of nonnegative integers. There is an operation that performs the following two steps on the grid:

Step 1. It chooses two numbers adjacent vertically or horizontally in the grid.
Step 2. It decreases each number chosen in Step 1 by 1 if it is positive.

For example, Figure 1 shows the result of four successive executions of the operation on a 2-by-2 grid of four numbers.

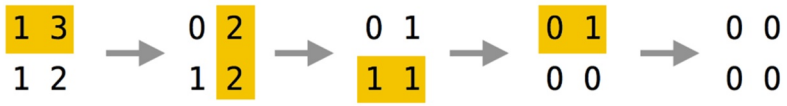


Figure 1. Four successive executions of the operation on a 2-by-2 grid of four numbers

In the above example, the operation was executed four times to make all the numbers in the initial grid to 0. You can easily check there is no way to do so using less number of executions of the operation.

You are to write a program that calculates the minimum number of executions of the operation to make all the numbers in the grid to 0.

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 ($2 \leq n, m \leq 50$), where n is the row size and m is the column size of a grid. In the following n lines, the elements of the grid are given row by row, that is, the i-th line contains m integers which correspond to the elements of the i-th row of the grid in order, for $1 \leq i \leq n$. Each integer in the grid is between 0 and 1,000, inclusively.

Output
Your program is to write to standard output. Print exactly one line for each test case. The line should contain an integer representing the minimum number of executions of the operations required in the problem.

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

Sample Input	Output for the Sample Input
2 2 2 1 3 1 2 2 4 2 3 2 3 1 2 1 1	4 8

The 40th Annual
ACM International Collegiate Programming Contest
Asia Regional - Daejeon

Problem D
Kernel
Time Limit: 0.1 Seconds

A farmer Will has a plan to purchase a robot management system for his farm. The system operates multiple autonomous robots in the farm to collect the useful information. To control the robots, the system has a server, called *beacon*, at a fixed position in the farm. At the end of day, the robots should be returned to the beacon, and checked their status for the next day. For this, the beacon keeps sending a special signal to the robots. The robots immediately know from the signal where the beacon is, and try to reach the beacon by moving continuously toward the beacon.

More precisely, the farm is modeled as a simple rectilinear polygon P such that horizontal and vertical edges appear alternatingly along its boundary, its vertices are all distinct, and two edges have no intersections except at their end vertices. A robot as well as a beacon is represented as a point *in* P, that is, on the boundary of P or in the interior of P. A robot p now moves in P greedily to minimize its Euclidean distance to the beacon b as follows: p moves along the ray from p to b until it reaches b or hits the boundary of P. If it hits the boundary, then it may either still reduce its Euclidean distance to b by sliding on the boundary, or it cannot reduce the distance even to any direction. A point p in P is *attracted* by b in P (equivalently, b *attracts* p) if it eventually reaches to b in P by strictly decreasing its Euclidean distance to b.

Let us look at the Figure 1 below. In the left figure, a point p can move along the path marked with thick solid segments, and finally reach to the beacon b. In the right figure, it can reach to x, but it cannot move anymore because there is no direction from x to reduce its Euclidean distance to b.

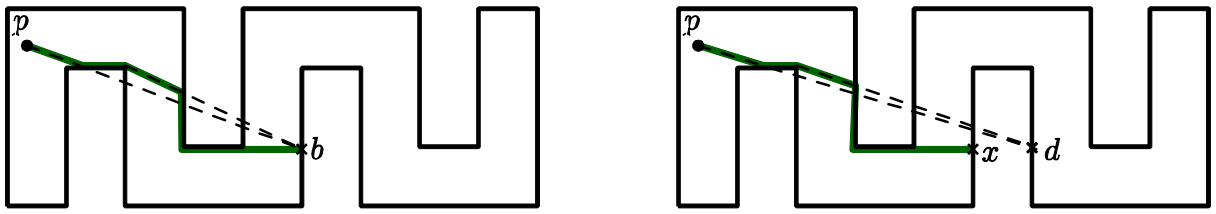


Figure 1. The cases where a beacon $\boldsymbol{b}$ can or cannot attract a point $\boldsymbol{p}$.

A *kernel* of P is the set of points in P that can attract all points in P. If a beacon is placed at any position of the kernel of P, then all points in P can be attracted by the beacon, so only one beacon is sufficient to call all the robots. But the kernel of some polygon may not exist.

Given a simple rectilinear polygon P of n vertices, write a program to decide whether its kernel exists or not.

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, n ($4 \leq n \leq 10{,}000$), where n is the number of vertices in a simple rectilinear polygon P. The following n lines give the coordinates of the vertices in counterclockwise direction. Each vertex is represented by two numbers separated by a single space, which are the x-coordinate and the y-coordinate of the vertex, respectively. Each coordinate is given as an integer between $-1{,}000{,}000{,}000$ and $1{,}000{,}000{,}000$, inclusively. Note that all the vertices are distinct.

Output

Your program is to write to standard output. Print exactly one line for each test case. The line should contain `YES` if the polygon given in the test case has a kernel, `NO` otherwise.

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

Sample Input	**Output for the Sample Input**
2 8 0 0 3 0 3 2 2 2 2 1 1 1 1 2 0 2 12 0 0 5 0 5 3 3 3 3 2 4 2 4 1 1 1 1 2 2 2 2 3 0 3	YES NO

The 40th Annual

ACM International Collegiate Programming Contest
Asia Regional - Daejeon

Problem E
Log Jumping
Time Limit: 0.1 Seconds

We want to arrange N logs in a circle for a playing ground shown in Figure 1. Kids are jumping one log to the next log circularly. We are concerned to arrange the logs to help kids by minimizing the height difference between two adjacent logs.

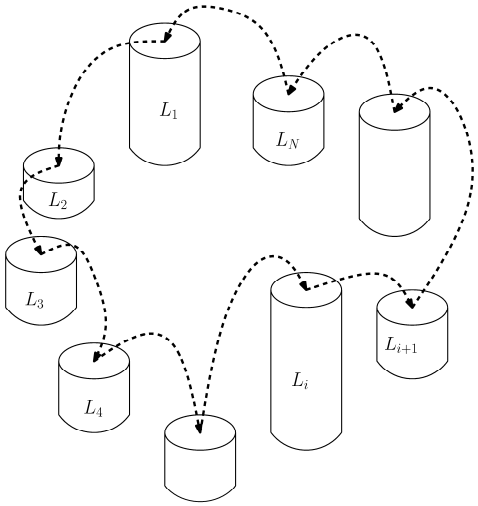


Figure 1. N logs are arranged in a circle.

The *difficulty* of log jumping depends on the maximum difference of heights among all pairs of two adjacent logs. Assume that we are given 5 logs with the heights {2, 4, 5, 7, 9}. Note that the last log with height 5 is immediate adjacent to the first log with height 2 in the arrangement [2, 9, 7, 4, 5]. The maximum height difference of the arrangement [2, 9, 7, 4, 5] is |2 – 9| = 7. We can make a better arrangement as [2, 5, 9, 7, 4] where the maximum height difference is |5 – 9| = 4. This arrangement [2, 5, 9, 7, 4] is optimal since there is no arrangement with the maximum height difference less than 4.

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, N ($5 \leq N \leq 10{,}000$), where N is the number of logs. In the next line, the heights of N logs, L_i are given as a sequence of integers. ($1 \leq L_i \leq 100{,}000$)

Output
Your program is to write to standard output. Print exactly one line for each test case. The line should contain an integer representing the maximum height difference of an optimal log arrangement. The following shows sample input and output for three test cases.

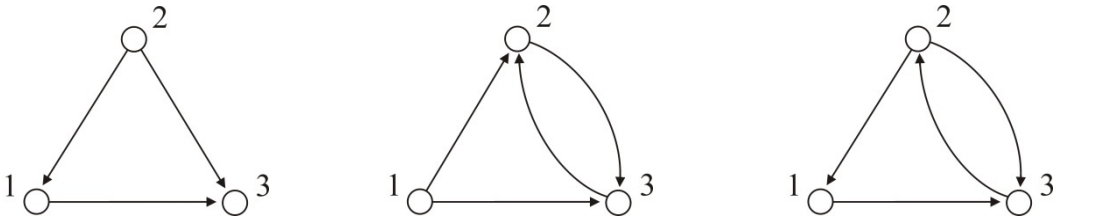
Sample Input	Output for the Sample Input
3 7 13 10 12 11 10 11 12 5 2 4 5 7 9 8 6 6 6 6 6 6 6 6	1 4 0

The 40th Annual
ACM International Collegiate Programming Contest
Asia Regional - Daejeon

Problem F
Odd Cycle
Time Limit: 1.5 Seconds

Recently, a smart student SeoJin realized that not a few hard problems for general undirected graphs are polynomially solvable for graphs that have no cycles of odd length. Wondering whether or not the same follows for directed graphs, she made up her mind to start a study on the subject under the supervision of HyeongSeok, a distinguished scholar in this field. Her first research finding is effective recognition of the directed graphs without odd cycles, i.e., an algorithm for determining if a given directed graph has a cycle of odd length. Still, she is promoting utmost efficiency in recognizing her class of graphs. In order to help her, you are going to write an ultra-efficient program that determines the existence of an odd cycle in a directed graph and report an arbitrary odd cycle, if any.

Let G be a simple directed graph having no self-loops or multiple edges. For any two vertices v and w of G, a (directed) path from v to w in G is a sequence $(u_1, u_2, \ldots, u_l)$ of distinct vertices of G such that $u_1 = v$, $u_l = w$, and u_i is adjacent to u_{i+1} in G for all $i \in \{1, \ldots, l-1\}$. If $l \geq 2$ and u_l is adjacent to u_1, the sequence is called a (directed) cycle. An odd cycle refers to a cycle of odd length, where the length of a cycle is simply the number of edges of the cycle. Refer to Figure 1 for illustrative examples.



(a) There is no cycle. (b) There is no odd cycle. (c) There is an odd cycle (2,1,3).

Figure 1. Simple directed graphs.

Input
Your program is to read from standard input. The input consists of T test cases, where the positive integer T is given in the first line of the input, followed by the description of each test case. The first line of a test case contains two positive integers n and m, respectively, indicating the numbers of vertices and directed edges in a simple directed graph, in which we assume $n \leq 100{,}000$ and $m \leq 1{,}000{,}000$. The vertices are indexed 1 to n. In the following m lines, each line contains two integers v and w, which represent a directed edge from vertex v to vertex w. The two integers given in a single line are always separated by a space.

Output
Your program is to write to standard output. For each test case, the first line must contain an integer indicating whether the given directed graph has an odd cycle. If yes, the integer must be `1`; otherwise `-1`. When and only when the first line is `1`, it must be followed by the description of an arbitrary odd cycle of the input graph. A cycle is described by a single line containing an integer l, representing its length, followed by l lines containing, one by one, the vertices encountered when we traverse the cycle starting from an arbitrary vertex. Note that the vertices of the cycle must be distinct.

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

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

The 40th Annual
ACM International Collegiate Programming Contest
Asia Regional - Daejeon

Problem G
Path
Time Limit: 0.5 Seconds

A *histogram* is a simple rectilinear polygon whose boundary consists of two chains such that the upper chain is monotone with respect to the horizontal axis and the lower chain is a horizontal line segment, called the *base* segment (See Figure 1).

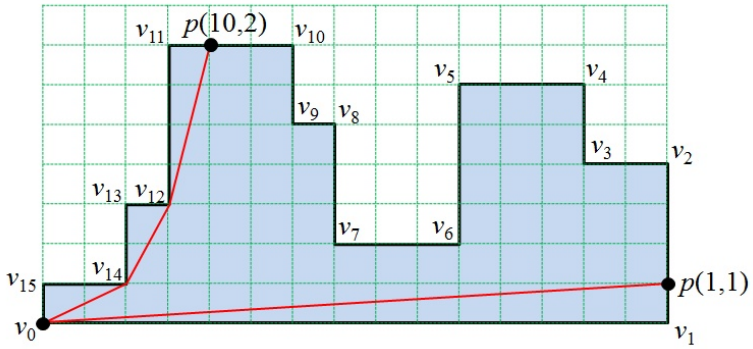


Figure 1. A histogram and its base segment (v_0, v_1)

Let P be a histogram specified by a list $(v_0, v_1, \ldots, v_{n-1})$ of n vertices in the counterclockwise order along the boundary such that its base segment is (v_0, v_1). An edge e_i is a line segment connecting two vertices v_i and v_{i+1}, where $i = 0, 1, \ldots, n-1$ and $v_n = v_0$.

A path inside P is a simple path which does not intersect the exterior of P. The length of the path is defined as the sum of Euclidean length of the line segments of the path. The *distance* between two points p and q of P is the length of the shortest path inside P between p and q. Your task is to find the distance between v_0 and each point of a given set S of points on the boundary of P. A point of the set S is denoted by $p(k, d)$ which represents a point q on the edge e_k such that d is the distance between v_k and q.

In the histogram of Figure 1, the shortest path between v_0 and $q_1 = p(10, 2)$ is a polygonal chain connecting v_0, v_{14}, v_{12} and q_1 in that order, and its length is 8.595242. The shortest path between v_0 and $q_2 = p(1, 1)$ is a segment directly connecting v_0 and q_2 with length 15.033296.

Given a histogram P with n vertices and a set S of m points on the boundary of P, write a program to find the distances between v_0 and all points of S.

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, n ($4 \le n \le 100{,}000$), where n is the number of vertices of a histogram $P = (v_0, v_1, \ldots, v_{n-1})$. In the following n lines, each of the n vertices of P is given line by line from v_0 to v_{n-1}. Each vertex is represented by two numbers, which are the x-coordinate and the y-coordinate of the vertex, respectively. Each coordinate is given as an integer between 0 and 1,000,000, inclusively. Notice that (v_0, v_1) is the base segment. The next line contains an integer m ($1 \le m \le 100{,}000$) which is the size of a set S given as your task. In the following m lines. Each point

$p(k, d)$ of S is given line by line, and is represented by two integers k and d, where $0 \leq k \leq n-1$ and $0 \leq d <$ the length of edge e_k. All points in the set S are distinct.

Output

Your program is to write to standard output. Print exactly one line for each test case. The line should contain exactly one real value which is the sum of the distances between v_0 and all points of S. Your output must contain the first digit after the decimal point, rounded off from the second digit. If each result is within an error range, 0.1, it will be considered correct. The Euclidean distance between two points $p = (x_1, y_1)$ and $q = (x_2, y_2)$ is $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$.

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

Sample Input	Output for the Sample Input
2 16 0 0 15 0 15 4 13 4 13 6 10 6 10 2 7 2 7 5 6 5 6 7 3 7 3 3 2 3 2 1 0 1 2 10 2 1 1 8 100000 100000 400000 100000 400000 200000 300000 200000 300000 300000 200000 300000 200000 200000 100000 200000 8 1 0 2 0 3 0 4 0 5 0 6 0 7 0 1 50000	23.6 1909658.1

The 40th Annual

ACM International Collegiate Programming Contest
Asia Regional - Daejeon

Problem H
Polynomial
Time Limit: 0.05 Seconds

A polynomial $f(k)$ of degree t with integral coefficients is given as $f(k) = c_0 + c_1k + c_2k^2 + \cdots + c_tk^t$, where the coefficients $c_0, \ldots, c_t$ are all integers. Here, we are interested in the sum $S(n)$ of $f(0)$, $f(1)$, …, and $f(n)$ for any nonnegative integer n. That is, $S(n)$ is defined by:

$$S(n) = \sum_{k=0}^{n} f(k) = f(0) + f(1) + \cdots + f(n).$$

The sum $S(n)$ is a polynomial, too, but is of degree $t + 1$ and rational coefficients. It can thus be represented by:

$$S(n) = \frac{a_0}{b_0} + \frac{a_1}{b_1}n + \frac{a_2}{b_2}n^2 + \ldots + \frac{a_{t+1}}{b_{t+1}}n^{t+1},$$

where a_i and b_i are integers that are relatively prime for each $i = 0,1, \ldots, t + 1$, or equivalently, that have no common divisor greater than 1.

Given a polynomial $f(k)$ of degree t with integral coefficients $c_0, \ldots, c_t$, your program is to compute $S(n)$ for the given polynomial $f(k)$ and to output the following value

$$\sum_{i=0}^{t+1} |a_i|,$$

where the a_i are determined as above for such a representation of $S(n) = \frac{a_0}{b_0} + \frac{a_1}{b_1}n + \frac{a_2}{b_2}n^2 + \ldots + \frac{a_{t+1}}{b_{t+1}}n^{t+1}$.

You may exploit the following known identity for polynomials: for any positive integer d and any real x,

$$(x + 1)^d - x^d = 1 + \binom{d}{1}x + \binom{d}{2}x^2 + \cdots + \binom{d}{d-1}x^{d-1},$$

where $\binom{d}{i} = \frac{d!}{i!(d-i)!}$ for any integer i with $0 \le i \le d$.

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 only a single line containing a nonnegative integer t $(0 \le t \le 25)$ and $t + 1$ following integers $c_0, \ldots, c_t$ with $-10 \le c_0, \ldots, c_t \le 10$ and $c_t \ne 0$. This fully describes the input polynomial $f(k) = c_0 + c_1k + c_2k^2 + \cdots + c_tk^t$ of degree t with coefficients $c_0, \ldots, c_t$.

Output
Your program is to write to standard output. Print exactly one line for each test case. The line should contain an integer representing the value $\sum_{i=0}^{t+1}|a_i|$.

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

Sample Input	Output for the Sample Input
3 3 1 1 1 1 5 0 -1 0 1 0 -1 5 -3 10 9 2 -7 5	17 6 206

The 40th Annual

ACM International Collegiate Programming Contest
Asia Regional - Daejeon

Problem I
Stock
Time Limit: 1 Second

Mr. Lim enjoys investing in stocks. His investment rules are as follows: Given the stock prices of a company for a series of days, each day his action is one of the followings:

(1) Buy one unit of stock.
(2) Sell any number of stock units you have already bought.
(3) Do nothing.

He wants to know the difference between the profit he obtained and the maximum possible profit by planning the above actions optimally. So, he is trying to calculate the maximum profit that can be obtained by an optimal plan for given the stock prices for a series of days. For example, let the stock prices of three days be 10, 7, and 6. Then the maximum profit is 0 because the price decreases each day. If the stock prices of three days are 3, 5, and 9, the maximum profit is 10 obtained by buying one unit of stock at each of the first two days and selling two stock units at the third day.

You are to write a program calculating the maximum profit that can be obtained by an optimal plan.

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, n ($2 \leq n \leq 1{,}000{,}000$), where n is the number of days in a period. In the following line, n integers representing stock prices of the period are given. Each stock price is given as an integer between 1 and 10,000, inclusively.

Output
Your program is to write to standard output. Print exactly one line for each test case. The line should contain the maximum profit. You may assume that the result fits in a signed 64-bit integer.

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

Sample Input	Output for the Sample Input
3 3 10 7 6 3 3 5 9 5 1 1 3 1 2	0 10 5

The 40th Annual

ACM International Collegiate Programming Contest
Asia Regional - Daejeon

Problem J
3-Primes Problem
Time Limit: 0.05 Seconds

In number theory, the 3-primes problem states that:

> *Every odd number greater than 5 can be expressed as a sum of exactly three primes. (A prime may be used more than once in the same sum.)*

Some examples of the problem are:

$$7 = 2 + 2 + 3$$
$$11 = 2 + 2 + 7$$
$$25 = 7 + 7 + 11$$

In 1939, Russian mathematician I. M. Vinogradov showed that any *sufficiently large* odd integer can be expressible as a sum of three primes. Later it is shown that sufficiently large in Vinogradov's proof meant numbers greater than $3^{3^{15}} \approx 10^{7000000}$. The best known improved bound for the figure is approximately $e^{3100} \approx 2 \times 10^{1346}$. This number is too large to admit checking all smaller numbers by computer.

Given a positive odd integer greater than 5, write a program to test whether or not the integer can be represented as a sum of exactly three primes, where the primes may not be distinct.

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 a positive integer $K (7 \leq K < 1{,}000)$.

Output
Your program is to write to standard output. Print exactly one line for each test case. Print three primes, in nondecreasing order, if the input number *K* can be represented as a sum of exactly three primes, otherwise print `0`(zero). If there are more than one case for three primes, print any case of them.

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

Sample Input	Output for the Sample Input
3 7 11 25	2 2 3 2 2 7 5 7 13

The 40th Annual
ACM International Collegiate Programming Contest
Asia Regional - Daejeon

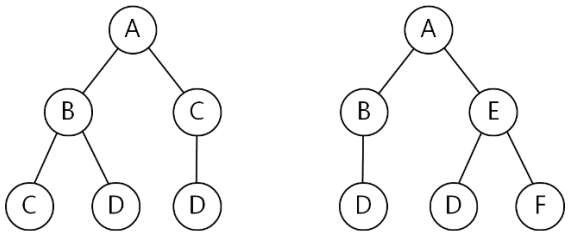
Problem K
Tree Edit
Time Limit: 0.5 Seconds

XML(e**X**tensible **M**ark-up **L**anguage) is a markup language that defines a set of rules for encoding documents. XML has come into common use for the interchange of data over the Internet. XML documents represent hierarchically structured data and are generally modeled as ordered labeled trees. Each node in such a tree represents an XML element and is labeled with a corresponding tag name. Each edge in this tree represents a relation between the child element and parent element in the corresponding XML document. Finding structural similarities among XML documents is one of central issues in information retrieval. Here, we consider the structural similarity of two XML documents represented as ordered labeled trees.

Let T be a rooted tree with one or more nodes. We call T a *labeled tree* if each node is assigned a label. The labels of a labeled tree are not necessarily distinct. We call T an *ordered tree* if a left-to-right order among siblings in T is given. Given two ordered labeled trees T_1 and T_2, the similarity of T_1 and T_2 is often measured by *Tree Edit Distance*, $TED(T_1, T_2)$. $TED(T_1, T_2)$ is defined as the minimum number of edit operations that transform T_1 to T_2. Three edit operations that can be applied to a given tree T are as follows:

(1) $Insert(x)$: Insert a leaf node whose label is x. After this operation, node x becomes an i-th child of its parent for some i, $1 \le i \le d+1$, if its parent had d children before this operation.
(2) $Delete(\,)$: Delete an arbitrary leaf node. This operation cannot be applied to a tree with a single node.
(3) $Relabel(x, y)$: Replace the label x of an arbitrary node by the label y.

Note that only one node is inserted, deleted, or relabeled by one edit operation. For example, let's consider two trees in Figure 1. If we apply the following three operations successively, then T_1 transforms to T_2: $Delete(\,)$, $Relabel(C, E)$, and $Insert(F)$, where the delete operation is applied to a leaf node C in T_1. By applying two or less operations, you cannot transform T_1 to T_2. So, $TED(T_1, T_2)$ is 3.



(a) T_1 (b) T_2

Figure 1. Two ordered labeled trees

You are to write a program calculating the tree edit distance of given two ordered labeled trees.

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 two lines. The first line contains a string

representing T_1 and the second line contains a string representing T_2. The representation of a tree is as follows: a tree consisting of a single root node with a label l is represented as (l); a tree consisting of a root labeled l with subtrees $T_1, T_2, \ldots, T_d$, is represented as $(l(r_1)(r_2)\cdots(r_d))$, where $(r_1), (r_2), \ldots, (r_d)$ are representations of $T_1, T_2, \ldots, T_d$, respectively. Each label of nodes is given as one uppercase character in the English alphabet. In the representation of a tree, there are no spaces and the number of nodes is between 1 and 1,000, inclusively.

Output

Your program is to write to standard output. Print exactly one line for each test case. The line should contain the minimum number of edit operations to transform T_1 into T_2.

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

Sample Input	Output for the Sample Input
3 (A(B(C)(D))(C(D))) (A(B(D))(E(D)(F))) (X(Y(Z(W)))) (W) (P(P)(P)(P)(P)(P)) (Q(Q)(Q)(Q)(Q))	3 4 6

The 40th Annual
ACM International Collegiate Programming Contest
Asia Regional - Daejeon

Problem L
Wheel of Numbers
Time Limit: 0.01 Seconds

A new gambling machine called "the wheel of numbers" is introduced. It is made up with a large wheel, which is equally divided into n segments. Each segment contains a digit between 0 and 9, inclusively. Initially you are given two m-digit numbers X and Y ($X \leq Y$). Note that these numbers may begin with 0. Then, you shoot a ball aiming at the fast-rotating wheel. Your ball will hit one of its segments. Then, you take m consecutive segments clockwise from there and obtain an m-digit number Z. You win if $X \leq Z \leq Y$ and lose otherwise.

For example, assume that the wheel is divided into $n = 8$ segments, and let [3, 7, 8, 3, 1, 9, 2, 7] be the numbers by picking one segment and reading numbers clockwise. If $X = 200$ and $Y = 311$, you win if your ball hits the segment with 2 as your number $Z = 273$ and surely $X = 200 \leq 273 \leq 311 = Y$.

Once you are given the wheel, X and Y, you would like to know your winning probability. It can be calculated easily if you know how many times the numbers satisfying the above mentioned condition appear on the wheel. Write a program that counts the number of their occurrences.

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 ($1 \leq n \leq 100$) and m ($1 \leq m \leq 9$ and $m \leq n$), where n is the number of segments in the wheel and m is the length of X and Y. The next line contains m digits representing X where each digit is between 0 and 9, inclusively, and separated by a space. The next line contains m digits representing Y as mentioned above. The following line contains n digits, separated by a space, obtained by reading numbers clockwise starting at some segment of the wheel. Again each digit is between 0 and 9, inclusively.

Output

Your program is to write to standard output. Print exactly one line for each test case. The line should contain an integer representing the number of occurrences of m-digit number Z contained in the wheel such that $X \leq Z \leq Y$. Note that if the same number appears more than twice, count them separately. For example, if 123 is the only number between X and Y and it appears twice on the wheel, the answer is 2, not 1.

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

Sample Input	**Output for the Sample Input**
3 8 3 2 0 0 3 1 1 3 7 8 3 1 9 2 7 5 2 8 8 9 9 1 3 2 5 4 6 3 0 0 0 9 9 9 1 2 3 4 5 6	1 0 6