2001

acm Asia regional - Taipei
International Collegiate Programming Contest

Problem Set

National Taiwan Normal University

- Problems: There are 8 problems (25 pages in all) in this packet.
- Program Input: Input to each program are done through the input file. Input filenames are given in the Problem Information Table.
- Program Output: All output should be directed to standard output (screen output).
- Time Limit: The judges will run each submitted program with certain time limit as given in the Problem Information Table.
- Judging: Please submit executable program for judging. Do not submit source file or test data file.
- Special note to java programmers: The first class in your Java program must be named, "hello", which should be compiled into "hello.class" for judging.

Table 1: Problem Information Table

	Problem Name	Input File	Time Limit
Problem A	Good Approximation Problem	pa.dat	30 secs.
Problem B	Power Cable Problem	pb.dat	30 secs.
Problem C	A Rate-Monotonic Scheduler	pc.dat	30 secs.
Problem D	The Constraint Densest Submatrix	pd.dat	30 secs.
Problem E	Hinge Node Problem	pe.dat	30 secs.
Problem F	Area of Simple Polygons	pf.dat	30 secs.
Problem G	Model Checking	pg.dat	30 secs.
Problem H	Composition of Functions	ph.dat	30 secs.

Problem A

Good Approximation Problem

Input File: pa.dat

It is well known that $\pi \approx \frac{22}{7}$. You can verify that for all integers p and q satisfying $1 \le q \le 7$ and $\frac{p}{q} \ne \frac{22}{7}$, we have $|22 - 7\pi| < |p - q\pi|$. Furthermore, $\frac{355}{113}$ is another good approximation of π. You can also verify that for all integers p and q satisfying $1 \le q \le 113$ and $\frac{p}{q} \ne \frac{355}{113}$, we have $|355 - 113\pi| < |p - q\pi|$.

Let p, q, x, y, x_1 and y_1 be integers, α be a real number and $q > 0$. We say that $\frac{y}{x}$ is *d-closer* to α than $\frac{y_1}{x_1}$ if $|y - x\alpha| < |y_1 - x_1\alpha|$. Notice that if α is also a rational number $\frac{p}{q}$, then the inequality $|yq - xp| < |y_1q - x_1p|$ is equivalent to $|y - x\alpha| < |y_1 - x_1\alpha|$. This can be used to avoid floating point operations. If $\frac{y}{x}$ is d-closer to α than any other rational number $\frac{y_1}{x_1}$ with denominator x_1 in the range from 1 to x, then $\frac{y}{x}$ is called a *good approximation* of α. Let $G(\alpha)$ be the set of all good approximations of α and $|G(\alpha)|$ be the cardinality of $G(\alpha)$. The *cardinality* of a set $G(\alpha)$ is the number of elements in $G(\alpha)$. We use an example to illustrate these symbols. Let $\alpha = \frac{37}{13}$. The good approximations of α are $\frac{3}{1}, \frac{17}{6}$ and $\frac{37}{13}$. The rational number $\frac{3}{1}$ is a good approximation of α since no other rational number with denominator 1 and an integer numerator is d-closer to α than $\frac{3}{1}$. The rational number $\frac{17}{6}$ is a good approximation of α since no other rational number with denominator in the range from 1 to 6 and an integer numerator is d-closer to α than $\frac{17}{6}$. A similar reason holds for $\frac{37}{13}$. It is clear that no rational number with denominator greater than 6 and an integer numerator is d-closer to α than $\frac{37}{13}$ since $|37 - 13\alpha| = 0$. Therefore, $G(\alpha) = \{\frac{3}{1}, \frac{17}{6}, \frac{37}{13}\}$ and $|G(\alpha)| = 3$. Similarly, you can find that $G(\frac{237}{113}) = \{\frac{2}{1}, \frac{21}{10}, \frac{65}{31}, \frac{86}{41}, \frac{237}{113}\}$ and $|G(\frac{237}{113})| = 5$.

Given a rational number α, you are asked to design a program for finding $|G(\alpha)|$.

Input: The first line of the input file contains an integer n, $n \le 5$, which represents the number of test cases. Then, the cases are listed line by line. In each line, there are two integers p_k and q_k separated by a space which are the numerator and denominator, respectively, of test case k, $k = 1, 2, \cdots, n$. Note that $1 \le p_k, q_k \le 10000$.

Output: List the value of $|G(\frac{p_k}{q_k})|$ in line k for $k = 1, 2, \cdots, n$.

Sample input

```
4
37  13
237  113
175   29
1432  6578
```

Output for the Sample Input

```
3
5
2
10
```

Problem B

Power Cable Problem

Input File: pb.dat

The downtown of city T consists of N, $1 \leq N \leq 10000$, tall commercial buildings that have basements. The buildings are numbered from 0 through $N-1$. The electricity of each building is provided by the City Electrical Power Company that puts all of its M, $1 \leq M \leq 1000$, power cables underground. In order for a building to have electricity, a power line must be connected from one of the underground cables to a power converter inside the building. Because of technical reasons, each power cable is a *loop*, meaning that it is a long cable line that originates from a mini power station, runs through some regions in the city and then comes back to the same power station. It is known that each power cable connects to at least 2 and at most 500 buildings. A building may be connected to zero, one or more than one power cable. The electricity of a building connected to more than one power cable can be provided by any one power cable by properly setting its power converter. To have a better city view, it is required by the law that power converters can only be built inside the basements.

During a *Typhoon*, the local rain storm, the downtown of city T is flooded. The basements of K, $1 \leq K \leq N$, buildings are filled with water. Fortunately, none of the mini power stations are damaged. Once a basement is flooded with rain water, its power converter is damaged and the building is out of electricity. Before fixing the power converter, we need to drain the water in the basement, which takes at least a long time. To make the situation worse, the power cables of city T are designed with the constraint that for each power cable, if it is connected with a damaged power converter, then none of the power converters connected to this power cable can be turned on. It is also impossible to disconnect the damaged power converts from the power cables. However, it is possible to properly set a power convert to get electricity from a power cable that carries electricity. After Typhoon, the City Electrical Power Company needs to know the total number of buildings that are out of electricity. Since the flood has made the traffics inside the city bad, the company cannot send people to survey. Fortunately, it is known by the company the buildings that are flooded in Typhoon since people from those buildings telephoned the company for help. Giving the original power line connection floor plans and the buildings that are flooded, your task is to calculate the total number of buildings that are currently out of electricity, including the ones that are originally not connecting to any power cable.

For example, each circle in Figure 1 represents a building. Two concentric circles represents a flooded building. There are 9 buildings. Buildings 7 and 8 are flooded. Solid straight lines are power cables. There are 3 power cable lines. One connects buildings 0, 1 and 6. One connects buildings 1, 2, 3 and 7. The last one connects buildings 0, 1, 4, 5 and 8. Buildings 2, 3, 4, 5, 7 and 8 do not have electricity currently in this example.

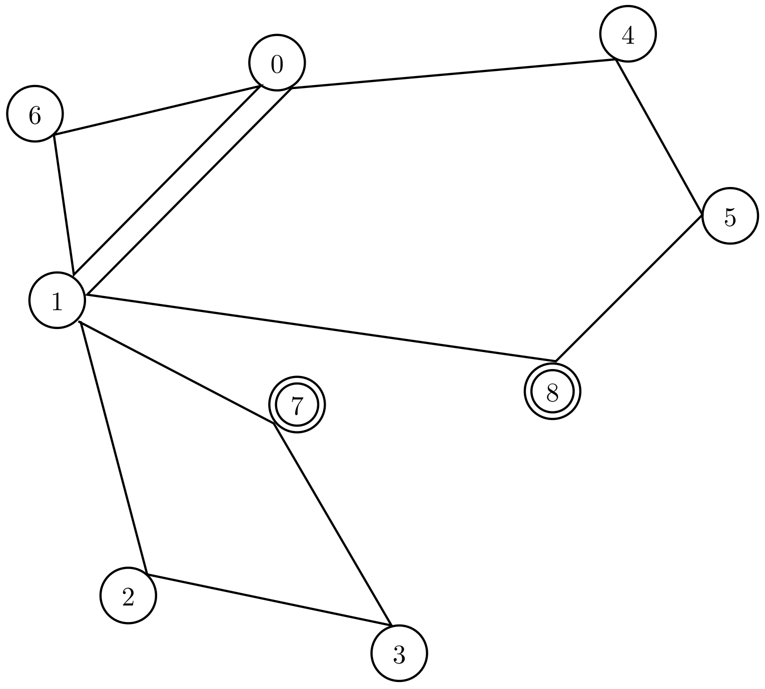

Figure 1: A power cable network.

Input: The input file consists of several test cases. In each test case, the first line consists of three integers N, M and K separated by a single space. Each of the following M lines represents in a power cable by beginning with the number of buildings in this power cable and then a list of buildings in this cable in clockwise order. It then followed by a line of K integers, each separated by a space, representing the buildings that are flooded. A line with three 0's separates two test cases. The end of the file is a line with three -1's.

Output: For each test case, output the total number of buildings that are out of electricity in a line.

Sample Input

```
9 3 2
3 0 1 6
4 1 7 3 2
5 0 4 5 8 1
7 8
0 0 0
5 2 1
3 0 2 1
3 1 4 3
4
-1 -1 -1
```

Output for the Sample Input

```
6
2
```

Problem C
A Rate-Monotonic Scheduler
Input File: pc.dat

Priority-driven scheduling approach is commonly used in modern computer operation systems where the systems always execute the task with the highest priority. In priority-driven scheduling, a task with lower priority may be preempted by a ready task with a higher priority and resume later. A *ready task* is a task ready to run as long as it gets the right to use CPU. A *periodic task* is executed exactly once in every constant interval which is called a *period*. For simplicity, we assume a periodic task is ready at the beginning of a period and the deadline is at the end of each period. It will be ready again at the beginning of the next period. Static-priority assignment states that the priorities of tasks are assigned in advance and do not change during task execution. A set of periodic tasks is said to be *feasible* if every task finishes execution before its deadline. *Rate-Monotonic algorithm* assigns a higher priority for a task with shorter period and is an optimal scheduling algorithm using static-priority assignment. It can also find a feasible schedule if there exists any scheduler using static-priority assignment that can find one. Given a set of periodic tasks with known constant execution times and periods, we are interested to know whether the set of task is feasible by a Rate-Monotonic scheduler and how many times the tasks are preempted before the schedule first repeats at the *hyperperiod* which is defined to be the least common multiple of all task periods. Note that it may be too large to calculate the hyperperiod, $< 2^{64}$, in advance and too slow to check feasibility on every time unit. It would be easier and faster to simulate the problem using two priority queues, one for the ready and preempted tasks and the other for the tasks not ready. All tasks will become ready again at the beginning of next hyperperiod.

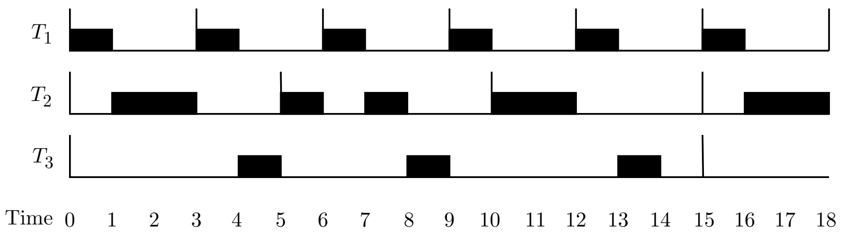

Figure 2: Task schedule example.

Figure 2 shows that tasks T_1, T_2, T_3 with execution times 1, 2, 3 and periods 3, 5, 15 respectively are feasible using the Rate-Monotonic scheduling algorithm since T_1, T_2, T_3 finish execution at time 1, 3, 14 respectively. T_2 is preempted by T_1 at time 6 and resume at time 7, T_3 is preempted by T_2 at time 5 and resume at time 8, preempted again by task T_1 at time 9 and resume at time 13. Therefore, the total number of preemption times for this task set is 3.

Input: All the input numbers are positive integers, < 500000, separated by a space or new line. The first line is the number of task sets. Then, the task sets are listed set by set. All the task sets are feasible. Each task set is listed by a line of the number of tasks, ≤ 10, and lines of task execution time and period pairs, execution time $<$ period. The periods are not sorted and are different in a task set.

Output: For each task set, each line shows the hyperperiod of the task set followed by the number of total preemption times.

Sample Input

```
3
3
1 3
2 5
3 15
2
1 2
1 3
2
123 123456
456 456123
```

Sample Output

```
15 3
6 0
18770373696 151
```

Problem D
The Constraint Densest Submatrix
Input File: pd.dat

Given an $m \times n$ matrix $A = [a_{ij}]$, the objective of the problem is to find a (consecutive) submatrix of A with size at least R rows and at least C cloumns such that the average value of the numbers in the submatrix is maximized.

Let $A = [a_{ij}]$ be an $m \times n$ matrix. Then a $p \times q$ matrix $B = [b_{ij}]$ is a (consecutive) submatrix of A if there exists a fixed ordered pair (k, ℓ) such that $b_{ij} = a_{i+k,j+\ell}$ for each (i, j) pair; note that $1 \le i \le p$ and $1 \le j \le q$. For example,

$$[\,7\,], \begin{bmatrix} 3 \\ 9 \end{bmatrix}, \begin{bmatrix} 1 & 9 \\ 7 & 3 \end{bmatrix} \text{ are submatrices of } \begin{bmatrix} 2 & 3 & 5 \\ 1 & 9 & 6 \\ 7 & 3 & 4 \end{bmatrix}$$

but

$$[\,8\,], [\,2 \quad 5\,], \begin{bmatrix} 9 & 6 \\ 7 & 3 \end{bmatrix} \text{ are not submatrices of } \begin{bmatrix} 2 & 3 & 5 \\ 1 & 9 & 6 \\ 7 & 3 & 4 \end{bmatrix}$$

The *density* of an $m \times n$ matrix $A = [a_{ij}]$ is the average value of all elements of A. That is, $(\sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij})/(mn)$. Note that finding the densest submatrix of a given matrix is easy. It is just the largest element within the matrix. On the other hand, the problem becomes interesting when we are asked to find the submatrix with at least R rows and at least C columns such that the density of the submatrix is maximized.

Write a program to find the densest submatrix of A with size *at least* $R \times C$. A naive method of calculating the densest submatrix of A considers $O(mn)$ possible upper left corner elements, together with another $O(mn)$ possible lower right corners, while calculating the density of the proposed submatrix takes $O(mn)$ time. What resulted is a very time-consuming $O(m^3n^3)$ time algorithm. For a modest sized 100×100 matrix, the term becomes 10^{12}, not easily finished in reasonable time. A somehow cleverer way of computing the densest submatrix is expected.

Given an $m \times n$ matrix $A = [a_{ij}]$, we denote (x, y)-*prefix sum* of A by the sum $P(x, y) = \sum_{i=1}^{x}\sum_{j=1}^{y} a_{ij}$. It follows that the sum of submatrix with upper left corner (i_1, j_1) and lower right corner (i_2, j_2), can be quickly computed by 4 prefix sums; that is,

$$\sum_{i=i_1}^{i_2}\sum_{j=j_1}^{j_2} a_{ij} = P(i_2, j_2) - P(i_1 - 1, j_2) - P(i_2, j_1 - 1) + P(i_1 - 1, j_1 - 1)$$

Input

Several sets of integral matrices. The inputs are just a list of integers. Within each set, the first 4 integers (in a single line) represent the size of the matrix, m and n, indicating an $m \times n$ matrix, and the constrained submatrix size, R and C. Note that each of them (m, n, R, C), can be as large as 200. After the four integers, there will be m lines representing the m rows of the matrix; each line (row) contains exactly n integers which are the elements in the row. The value of each element in a matrix is in the range from 0 to 800 and most of them are less than 100. Thus, there are totally mn integers for the particular matrix.

These matrices will occur repeatedly in the input as the pattern described above. An integer $m = 0$ (zero) signifies the end of input.

Output

For each matrix of the input, find the densest submatrix with size *at least* R rows and C columns. Output the submatrix by specifying the upper left corner and the lower right corner by printing four indices. For example, a line

$$r_1 \quad c_1 \quad r_2 \quad c_2$$

represents a submatrix with the upper left corner (r_1, c_1) and the lower right corner (r_2, c_2). Output a single star '$*$' to signify the end of outputs.

For the uniqueness of the answer, if two submatrices have the same density, only print the matrix whose four indices of corners (r_1, c_1, r_2, c_2) with smaller *lexicographical order*. For example, if the two sets of indices are $(4, 3, 18, 9)$ and $(7, 1, 14, 8)$, then just output the first submatrix since its indices of the corners has smaller lexicographical order.

Sample Input

```
  3    4     2    1
150  500   150  800
  1  200   100  300
400  800    80  400
  4    2     3    2
400  800
200  500
100  200
600  600
  0
```

Sample Output

```
1  4  2  4
1  1  4  2
*
```

Problem E

Hinge Node Problem

Input File: pe.dat

In decades, people have realized the significance of data communication. Most of the designs and analysis of communication networks usually model their topologies as graphical representations because many relevant problems of networks can be solved by using graph theoretic results. As usual, a communication network is modeled by a graph that nodes and edges in a graph correspond to the communication sites and links, respectively. A network $G = (N, E)$ consists of a set N of nodes together with a set E of edges, representing pairs of nodes. If the pairs are considered to be unordered, then we have an *undirected* network and the edge joining two nodes u and v is represented by (u, v). For example, Figure 3 depicts a network G which contains 10 nodes and 16 edges.

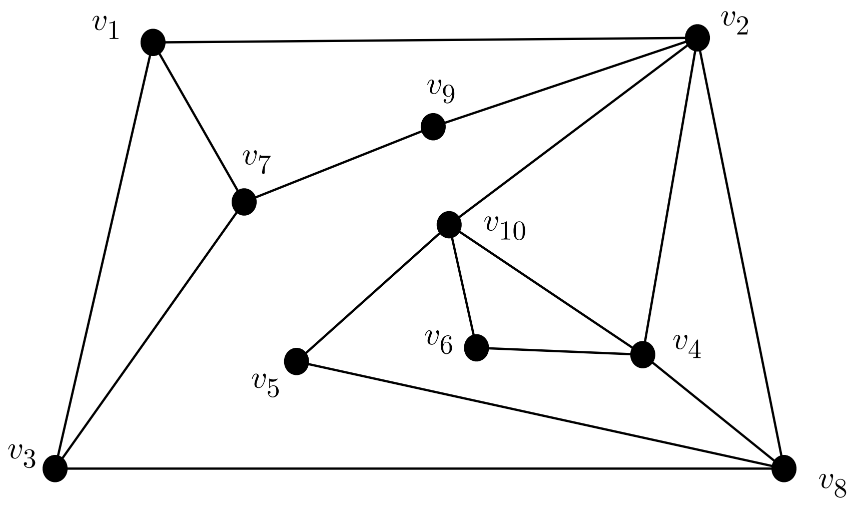

Figure 3: A network G.

In a network G, the *distance* between two nodes u and v, denoted by $d_G(u, v)$, is the number of edges of a shortest path from u to v in G. A sequence of vertices $v_1, v_2, \cdots, v_k$ is a *path* from v_1 to v_k of *length* $k - 1$ in G provided that there is an edge between v_i and v_{i+1} for $i = 1, 2, \cdots, k - 1$. If no path exists between nodes u and v, then $d_G(u, v) = \infty$. A path is a *shortest path* between nodes u and v if its length is minimum among all of the paths between u and v. A network is *connected* if there exists a path between any two nodes. The failure of a node w means that w and all its incident edges are removed from G, and the remaining subnetwork is denoted by

$G-w$. A node w is called a *hinge node* if there exist two other nodes u and v in G such that $d_{G-w}(u,v) > d_G(u,v)$. It means that the distance between u and v is increased after w is removed from G. Thus, a hinge node can be viewed as a critical node of the corresponding network and the failure of such a node will increase the communication cost to the remaining subnetwork. For example, we consider the network G in Figure 3. The node v_2 is a hinge node since $d_{G-v_2}(v_8,v_9) = 3 > d_G(v_8,v_9) = 2$. Indeed, the set of hinge nodes contained in G is $\{v_2, v_3, v_4, v_7, v_8, v_{10}\}$.

Suppose that we have several networks. Each network is connected and contains at most n nodes, where $3 \leq n \leq 100$. Assume now that you are hired to serve as a network administrator and you should analyze the communication cost. For this reason, you will be interested in finding all hinge nodes in a network. In particular, you should design a program that can efficiently calculate the total number of hinge nodes for each of the given networks.

Input: The input file consists more than one and less than six networks (cases). Each test case starts with a positive integer n, where $3 \leq n \leq 100$. The following n lines represents the adjacency matrix of a network G. The last case is followed by a "0" to indicate "end of input." An *adjacency matrix* of a network G with n nodes, denoted by $A(G) = [a_{u,v}]$, is an $n \times n$ $0,1$-matrix such that $a_{u,v} = 1$ if $(u,v) \in E$, and $a_{u,v} = 0$ otherwise. Note that there is not any delimiter between any two elements in each line of a $0,1$-matrix. For example, the adjacency matrix of the graph in Figure 3 is shown in test case 3 of the sample input.

Output: For each test case, output the total number of hinge nodes in a line.

Sample Input

```
3
010
101
010
3
011
101
110
10
0110001000
1001000111
1000001100
0100010101
0000000101
0001000001
1010000010
0111100000
0100001000
0101110000
0
```

Output for the Sample Input

```
1
0
6
```

Problem F
Area of Simple Polygons
Input File: pf.dat

There are N, $1 \leq N \leq 1,000$ rectangles in the 2-D xy-plane. The four sides of a rectangle are horizontal or vertical line segments. Rectangles are defined by their lower-left and upper-right corner points. Each corner point is a pair of two non-negative integers in the range of 0 through 50,000 indicating its x and y coordinates. Assume that the contour of their union is defined by a set S of segments. We can use a subset of S to construct simple polygon(s). Please report the total area of the polygon(s) constructed by the subset of S. The area should be as large as possible. In a 2-D xy-plane, a polygon is defined by a finite set of segments such that every segment extreme (or endpoint) is shared by exactly two edges and no subsets of edges has the same property. The segments are edges and their extremes are the vertices of the polygon. A polygon is *simple* if there is no pair of nonconsecutive edges sharing a point. Notice that the area of a polygon includes the holes inside the polygon.

Example: Consider the following three rectangles:

$$\text{rectangle 1: } < (0,0)\ (4,4) >,$$
$$\text{rectangle 2: } < (1,1)\ (5,2) >,$$
$$\text{rectangle 3: } < (1,1)\ (2,5) >.$$

The total area of all simple polygons constructed by these rectangles is 18.

Input: The input consists of multiple test cases. A line of 4 -1's separates each test case. An extra line of 4 -1's marks the end of the input. In each test case, the rectangles are given one by one in a line. In each line for a rectangle, 4 non-negative integers are given. The first two are the x and y coordinates of the lower-left corner. The next two are the x and y coordinates of the upper-right corner.

Output: For each test case, output the total area of all simple polygons in a line.

Sample Input

```
0 0 4 4
1 1 5 2
1 1 2 5
-1 -1 -1 -1
0 0 2 2
1 1 3 3
2 2 4 4
-1 -1 -1 -1
-1 -1 -1 -1
```

Output for the Sample Input

```
18
10
```

Problem G
Model Checking
Input File: pg.dat

To build a complex system, software engineering methods suggest building a design or prototype first before programming the real system. System correctness can be tested or verified on such a design (or a model) so that serious errors can be found in an earlier stage. As a result, software development cost and time can be significantly reduced.

A design is often the abstract behaviors of the real system. Some representations, such as finite-state machines (FSM), are suitable for describing the abstract behaviors of a system. For example, if we want to build an oven system, we can first design a FSM in Figure 4 to represent the oven we intend to build. Then, we can check system correctness on the FSM.

To define the correctness, we label every state of the machine with propositions {**start**,**close**,**heat**}. Each proposition at a state is either true or false. For instance, at state 1, all propositions are false, which means at this state the start button is not pressed, oven door is not closed, and oven is not heated. At state 3, proposition **close** is true, which means at the state, oven door is closed.

An interesting problem is to ask if the FSM satisfies some properties. For example, we may want to ask "*Can the oven **never** heat up after start button is pressed?*" Let it be property A. If the FSM satisfies property A, we know the oven is a bad design and it must be fixed before being made into a product. To check if the FSM satisfies property A, it is equivalent to ask "*if there exist some states that* **start** *is true and* **heat** *is never true along an infinite run which begins from the states.*" In order to express the properties formally, we need the following notation.

An *infinite sequence* of states is a cycle or a path which finally goes into a cycle in a FSM. For example, see Figure 4. States 1, 2, 5 and 3 form a cycle and constitute an infinite sequence. States 7 and 4 constitute another infinite sequence since the path starts from state 7 and finally goes into the cycle which only contains state 4. Let $EG(p)$ denote the states which constitute some infinite sequences with proposition p being true along the sequences. For example, $EG(\sim$**heat**$)$ contains states 1, 2, 5 and 3. However, state 6 is not in $EG(\sim$**heat**$)$ since it cannot constitute an infinite sequence with other states satisfying p. $EG($**heat**$)$ contains states 7 and 4 which constitute an infinite sequence with **heat** being true along the sequence.

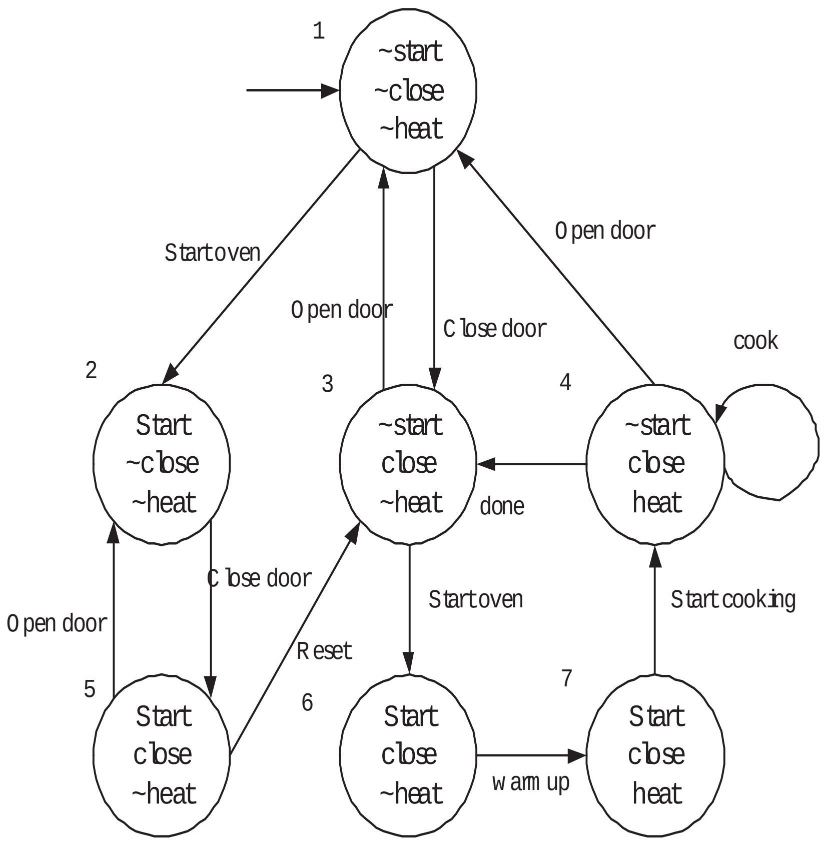

Figure 4: An oven's finite-state machine with propositions.

Therefore, property A can be expressed by a formula $(\textbf{start} \wedge EG(\sim \textbf{heat}))$ which contains states 2 and 5, where $\wedge$ is the "AND" relation of two propostions. The reason is as follows: The states which satisfy propostion **start** are 2, 5, 6 and 7. The stats which belong to $EG(\sim\textbf{heat})$ are 1, 2, 3 and 5. Thus, only states 2 and 5 satisfy the formula $(\textbf{start} \wedge EG(\sim \textbf{heat}))$.

Your goal. Given a FSM and a property formula, please write a program to determine if the FSM satisfies the property. If it can be satisfied, please output the states which satisfy the property.

Note that the property formula is limited to the form $(term_1 \wedge term_2 \wedge \ldots \wedge term_n)$, where each term is either a proposition p or $EG(p)$. Between terms in a formula, we always have the "AND" relation.

Input

The input begins with a number of test data which is less than 10. Each test data contains the following: A test data begins with the names of propositions. The first line begins with (m $prop_1$ $prop_2$ $\ldots prop_m$), where m, < 10, is the number of propositions, $prop_i$ is the name of ith proposition and each name contains less than 6 characters.

The second part is the information of a FSM. A FSM begins with (s t i) where $s < 50$ is the number of states, $t < 100$ is the number of transitions and $i < 100$ is the index of initial state. The index of states starts from 1. Next is s lines of data to describe the states. Each state is of the form (si p_1 p_2 $\ldots p_m$), where si is the index of the state and p_i is the boolean value of $prop_i$. It can be either T or F. After the state information is t lines of transition information. Each transition is described by (st ed), where st and ed are the source and destination states of the transition. Note that the label of transition is ignored because it is not a concern to our problem.

The final part is the property formula information. It is of the form (k $term_1$ $term_2 \cdots$ $term_k$) , where k is the length of the formula and $term_i$ can be either a proposition name $prop_i$ (with or without $\sim$) or $EG(prop_i)$.

Output

The states which satisfy the property formula. If there are no states satisfying the formula, please output “NO.”

Sample Input

```
2
3 start close heat
7 12 1
1 F F F
2 T F F
3 F T F
4 F T T
5 T T F
6 T T F
7 T T T
1 2
2 5
5 2
5 3
1 3
3 1
3 6
6 7
7 4
4 3
4 4
4 1
2 start EG~heat
2 a b
3 4 1
1 T T
2 F F
3 T F
1 2
2 3
1 3
3 3
2 EGa EG~b
```

Sample Output

```
2 5

3
```

Problem H
Composition of Functions
Input File: ph.dat

Professor T. T. Moh has invented a new public key cryptosystem. The new cryptosystem is called TTM, and it is based on the composition of functions.

TTM is a block cipher. Assume that the block size is 100 bytes. Thus, a block of message can be represented by $m = (m_1, m_2, \ldots, m_{100})$.

A sequence of invertible transformations $\phi_1, \phi_2, \ldots, \phi_n$ are designed to convert a block of message into cipher. Each transformation, ϕ_i, contains 100 functions, $\phi_i = (f_1^i, f_2^i, \ldots, f_{100}^i)$, where each f_j^i is a polynomial. Thus, each transformation can convert a block of 100 bytes into another block of 100 bytes.

After these functions are determined, the cipher $c = (c_1, c_2, \ldots, c_{100})$ can be computed by

$$c = \phi_n \circ \phi_{n-1} \circ \cdot \circ \phi_1(m).$$

The public key of the TTM cryptosystem is the composition of the transformations $\phi = \phi_n \circ \phi_{n-1} \circ \cdot \circ \phi_1$, and the private key is the transformations $\phi_1, \phi_2, \ldots, \phi_n$. The TTM cryptosystem is secure if the decomposition of functions is difficult,

Note that TTM does not treat a block of message as a large number. All computations in TTM is based on bytes. This is why the encryption and the decryption speed can be very fast.

To implement the TTM cryptosystem, we first need to select a field which has 2^8 elements. We shall denote this field by $\mathbf{F}(2^8)$, the addition operation by "+" and the multiplication operation by "·".

An element of $\mathbf{F}(2^8)$ can be represented by an 8-bit binary number $b_8 b_7 \cdots b_0$ or by a 2-digit hexadecimal number $h_1 h_0$.

To define "+" and "·" operation in $\mathbf{F}(2^8)$, we identify each element in $\mathbf{F}(2^8)$, $b_7 b_6 \cdots b_0$, with a polynomial

$$b_7 x^7 + b_6 x^6 + \cdots + b_0.$$

For example, 01010111 is the polynomial $x^6 + x^4 + x^2 + x + 1$.

The addition of two elements in $\mathbf{F}(2^8)$ is the addition of the two corresponding polynomials. For example 01010111 + 10000011 = 11010100, since $(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2$.

To define "$\cdot$" in $\mathbf{F}(2^8)$, we need an irreducible polynomial of degree 8. In this problem, we assume that the irreducible polynomial is

$$m(x) = x^8 + x^4 + x^3 + x + 1.$$

Multiplication of two elements in $\mathbf{F}(2^8)$ is the product of the two polynomials modular the irreducible polynomial $m(x)$. That is, first multiply the two polynomials and then divide the resulting polynomial by $m(x)$ and find the remainder. For example, $01010111 \cdot 10000011 = 11000001$, since $(x^6 + x^4 + x^2 + x + 1) \times (x^7 + x + 1) = x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$ and $x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$ modulo $x^8 + x^4 + x^3 + x + 1$ equals $x^7 + x^6 + 1$.

Note that, in the addition and multiplication of the polynomials, the operation of the coefficients is in $\mathbf{Z}_2$, not in the field $\mathbf{F}(2^8)$. That is, $0 + 0 = 1 + 1 = 0$, $0 + 1 = 1 + 0 = 1$, and $0 \times 0 = 0 \times 1 = 1 \times 0 = 0$, $1 \times 1 = 1$.

Although subtraction and division in $\mathbf{F}(2^8)$ are not needed in this problem, the following facts can help you better understand the operations of the field. It is easy to verify that 00000000 is the zero and 00000001 is the unit of the field $\mathbf{F}(2^8)$. Observe that subtraction in this field is the same as addition. They are all equivalent to the exclusive-or of the corresponding binary bits.

In this problem, we are going to design a simplified tool to generate keys for TTM. Your program needs only do two types of operations: (1) create a function, and (2) compute the composition of functions.

Creating a function means to read a function definition from the input file and store it properly. A function is defined in the input file as

$$\mathrm{f}i = t_1 + t_2 + \cdots + t_m.$$

In the above definition, i is the function number. Each function has a unique number. Each term t_j, $j = 1, 2, \ldots, m$ is a product of the form

$$c\ \mathrm{x}i_1\hat{}j_1\ \mathrm{x}i_2\hat{}j_2 \cdots \mathrm{x}i_r\hat{}j_r,$$

where c is an element in $\mathbf{F}(2^8)$, and each $\mathrm{x}i_k\hat{}j_k$, $k = 1, 2, \ldots, r$, means $(x_{i_k})^{j_k}$. The value of each i_k is an integer in $[1, 100]$. We shall adopt the following conventions in defining a term. If $c = 1$, the constant is omitted, except it is the only one in the term.

We assume that $j_k > 0$, and if the exponent $j_k = 1$, only xi_k will be shown. Note that the exponent j_k will be less than 256.

For example, "f1 = x1 + x2ˆ3 x3" and "f2 = 3f x10 x2 + x5ˆ7 + fe" are valid function definitions, but "f3 = 256 x1 + 320" is not valid. Constants must be in the range $[0, 255]$ and expressed by at most two hexadecimal digits. To make it simple, no parenthesis can be used in defining a function. For example, "f4 = (x1 + x2) x3" is not allowed.

The composition of functions is also considered as a definition of a function and it is expressed as

$$\mathrm{f}i = \mathrm{f}j(arguments)$$

where fj is a function define before, and the *arguments* is a list of functions f_{k_1}, f_{k_2}, ..., f_{k_r}. each function in the list must be define before. Each occurrence of x_l in the function fj will be substituted by f_{k_l} We need not list 100 functions, but enough functions must be listed to do the substitution.

Input

The input file contains a sequence of function definitions and function compositions. Each function definition and each function composition will be written in a line. A line will be very long, but no more than 900 characters.

Output

For each function definition, store the function properly. No outputs are needed for function definitions. For each function composition, compute and simplify the function and then print the results. You can either have a space or no space between terms. The polynomial printed must be simplified in "sum of product" format, without parenthesis, and two terms which differ in only the constants must be added together.

Sample Input

```
f1 = x1^2
f2 = x1 + x2
f3 = f1(f2)
f10 = x1^2 + x2^2
f11 = f10(f2, f2)
```

Output for the Sample Input

```
f3 = x1^2 + x2^2
f11 = 0
```