MIT Programming Contest
Team Round 3 Problems
2006

October 15, 2006

1 Match Broadcasting

A TV-network plans to broadcast an important football match. Their network of transmitters and users can be represented as a tree. The root of the tree is the transmitter that broadcasts the football match, the leaves of the tree are the potential users and other vertices in the tree are relays (transmitters). The price of transmission of a signal from one transmitter to another or to a user is given. A price of the entire broadcast is the sum of prices of all individual signal transmissions.

Every user is ready to pay a certain amount of money to watch the match and the TV-network then decides whether or not to provide the user with the signal.

Write a program that will find the maximal number of users able to watch the match so that the TV-network doesn't lose money from broadcasting the match.

Input

All data is read from the standard input. The first line of the input file contains two integers N and M, $2 \leq N \leq 3000, 1 \leq M \leq N-1$, the number of vertices in the tree and the number of potential users.

The root of the tree is marked with the number 1, while other transmitters are numbered 2 to $N - M$ and potential users are numbered $N - M + 1$ to N. The following $N - M$ lines contain data about the transmitters in the following form:

$$K \quad A_1 \quad C_1 \quad A_2 \quad C_2 \quad \ldots \quad A_K \quad C_K$$

Means that a transmitter transmits the signal to K transmitters or users, every one of them described by a pair of numbers A and C, the number of a transmitter or a user, and the cost of transmitting the signal to it.

The last line contains data about the users. It contains M integers representing respectively the price every user is willing to pay to watch the match.

Output

The only line of the output file should contain the maximal number of users able to watch the match, without the TV-network losing money.

Example

Sample Input	Sample Output
5 3 2 2 2 5 3 2 3 2 4 3 3 4 2	2

2 ID

T. Chur teaches various groups of students at university U. Every U-student has a unique Student Identification Number (SIN). A SIN s is an integer in the range $0 \leq s \leq MaxSIN$ with $MaxSIN = 10^6 - 1$. T. Chur finds this range of SINs too large for identification within her groups. For each group, she wants to find the smallest positive integer m, such that within the group all SINs reduced modulo m are unique.

Input

You should read all data from standard input.

On the first line of the input is a single positive integer N, telling the number of test cases (groups) to follow. Each case starts with one line containing the integer G ($1 \leq G \leq 300$): the number of students in the group. The following G lines each contain one SIN. The SINs within a group are distinct, though not necessarily sorted.

Output

You should write all data to standard output.

For each test case, output one line containing the smallest modulus m, such that all SINs reduced modulo m are distinct.

Example

Sample Input	Sample Output
2 1 124866 3 124866 111111 987651	1 8

This page was intentionally left blank.

3 Bundling

Outel, a famous semiconductor company, released recently a new model of microprocessor called Platinium. Like many modern processors, Platinium can execute many instructions in one clock step providing that there are no dependencies between them (instruction I_2 is dependent on instruction I_1 if for example I_2 reads a register that I_1 writes to). Some processors are so clever that they calcualte on the fly which instructions can be safely executed in parallel. Platinium however expects this information to be explicitly specified. A special amrker, called simply a *stop*, inserted between two instructions indicates that some instructions after the stop are possibly dependent on some instructions before the stop. In other words, instructions between two successive stops can be executed in parallel and there should not be dependencies between them.

Another interesting feature of Platinium is that a sequence of instructions must be split into groups of one, two or three successive instructions. Each group has to be packed into a container called a *bundle*. Each bundle has 3 slots and a single instruction can be put into each slot, however some slots may stay empty. Each instructions is categorized into one of 10 instruction types denoted by consecutive capital letters from A to J (instructions of the same type have similar functionality, for example type A groups integer arithmetic instructions and type F groups floating-point instructions). Only onstricutions of certain types are allowed to be packed into one bundle. A *template* specifies one permissible combination of instruction types within a bundle. A template can also specify a position of a stop in the middle of a bundle (there is at most one such stop allowed). In addition, stops are allowed between any two adjoining bundles. A set of templates is called a *bundling profile*. When packing instructions into bundles, one has to use templates from the bundling profile only.

Although Platinium is equipped with an instruction cache it was found that for maximal performance it is most crucial to pack instructions as densely as possible. A second important thing is to use a small number of stops.

Your task is to write a program for bundling Platinium instructions. For the sake of simplicity, we assume that the instructions cannot be reordered.

Write a program that reads a bundling profile and a sequence of instructions, computes the minimal number of bundles into which the sequence can be packed without breaking the dependencies and the minimal number of all stops that are required for the minimal number of bundles, then writes the result.

Input

The first line of the input contains two integers t and n separated by a single space. Integer t ($1 \leq t \leq 1500$) is the number of templates in the bundling profile. Integer n ($1 \leq n \leq 100000$) is the number of instructions to be bundled.

Each of the next t lines specifies one template and contains 3 capital letters t_1, t_2, t_3 with no spaces in between followed by a space and an integer p. Letter t_i (A$\leq t_i \leq$J) is an instruction type allowed in the i-th slot. Integer p ($0 \leq p \leq 2$) is the index of the slot after which the stop is positioned (0 means no stop within the bundle).

Each of the next n lines specifies one instruction. The i-th line of these n lines contains one capital letter c_i and an integer d_i, separated by a single space. Letter c_i (A$\leq c_i \leq$J) is the type of the i-th instruction. Integer d_i ($0 \leq d_i < i$) is the index of the last instruction (among the previous ones) that the i-th instruction is dependent on (0 means that the instruction is not dependent on any former instruction).

You can assume that for each instruction type c appearing in the instruction sequence there is at least one template containing c.

Output

The first and only line of the output contains two space-separated integers b and s, followed by a newline. Integer b is the minimal number of bundles in a valid packing. Integer s is the minimal number of all stops that are required for the minimal number of bundles.

Example

Sample Input	**Sample Output**
4 9 ABB 0 BAD 1 AAB 0 ABB 2 B 0 B 1 A 1 A 1 B 4 D 0 A 0 B 3 B 0	4 3

4 The Big Apple

Mebaa is very sad. He wants to live in New York City but is stuck living in some graph in the middle of nowhere. His hometown graph is an unweighted, undirected graph. One day he decides that he wants to see just how much like Manhattan he can make his hometown be. Let V denote the vertices in his graph and $d(u, v)$ be the shortest path distance between two vertices u, v in the graph. Let $\ell_1(x, y)$ be the Manhattan distance between two points x, y in $\mathbb{R}^k$. The Manhattan distance of $x, y \in \mathbb{R}^k$ is defined as

$$\sum_{i=1}^{k} |x_i - y_i|$$

He decides on the following notion of similarity. For a given function $f : V \to \mathbb{R}^k$ that does not shrink pairwise distances (i.e. $\forall\ u, v \in V\ d(u, v) \leq \ell_1(f(u), f(v))$, the *distortion* of f, $D(f)$, is the minimum r such that $\forall\ u, v \in V\ \ell_1(f(u), f(v)) \leq r \cdot d(u, v)$. If the graph is disconnected then $D(f) = \infty$ for any f.

Mebaa decides to pretend he's living in Manhattan by pretending vertices in the graph are actually points in $\mathbb{R}^k$. He wants to know the minimum possible distortion that can be achieved when mapping the vertices in his graph under the shortest path metric into $\mathbb{R}^k$ with the ℓ_1 metric. Of course, the minimum possible distortion you can achieve depends on k, the dimension of the points you map graph vertices into. For example, in a 4-node cycle, an embedding into 1 dimension cannot have distortion smaller than 2. However, distortion 1 is possible in 2 dimensions. You should tell Mebaa the minimum possible distortion over all possible dimensions he could map into. Thus, the correct answer for the 4-node cycle is 1.

Input

Each test case starts with a line N ($1 \leq N \leq 10$) containing the number of vertices in Mebaa's graph. The subsequent N lines each contain a string of N characters. Each character is a 0 or 1. The jth character of line i is a 1 if and only if there is an edge between vertices i and j. The jth character of line i will equal the ith character of line j. The ith position of line i will always be 0.

Output

Output the best possible distortion of the input graph into ℓ_1, followed by a newline. Your answer should be correct within $\pm 10^{-2}$. If the best possible distortion is infinite, output "INFINITY".

Example

Sample Input	Sample Output
4 0100 0010 0001 1000	1

5 Hang

Little Tom is learning how to program. He has just written some programs but is afraid to run them, because he does not know if they will ever stop. Please write a program to help him.

This task is not as easy as it may seem, because Tom's programs may behave nondeterministically.

Given a program written by Tom, your program should tell him whether his program can stop and if so, what is the shortest possible time before it stops.

Tom's computer consists of 32 1-bit registers and the program consists of n instructions. The registers are numbered from 0 to 31 and the instructions are numbered from 0 to $n-1$.

Below, `MEM`$[a]$ stands for the contents of the a-th register, $0 \leq a, b < 32,\ 0 \leq x < n,\ 0 \leq c \leq 1$.

Instruction	Semantics
`AND` a b	`MEM`$[a]$:= `MEM`$[a]$ and `MEM`$[b]$
`OR` a b	`MEM`$[a]$:= `MEM`$[a]$ or `MEM`$[b]$
`XOR` a b	`MEM`$[a]$:= `MEM`$[a]$ xor `MEM`$[b]$
`NOT` a	`MEM`$[a]$:= not `MEM`$[a]$
`MOV` a b	`MEM`$[a]$:= `MEM`$[b]$
`SET` a c	`MEM`$[a]$:= c
`RANDOM` a	`MEM`$[a]$:= random value (0 or 1)
`JMP` x	jump to instruction number x
`JZ` x a	jump to instruction number x if `MEM`$[a] = 0$
`STOP`	stop the program

The last instruction of every program is always `STOP` (although there can be more than one `STOP` instruction in a program). Every program starts with the instruction number 0. Before the start, the contents of the registers can be arbitrary values. Each instruction (including `STOP`) takes 1 processor cycle to execute.

You should read in the program, compute its shortest possible running time, then write the result.

Input

The first line of the input contains an integer n ($1 \leq n \leq 16$) being the number of instructions. Each of the next n lines contains one instruction of the program in the format given above. You may assume that the only whitespace characters in the program are single spaces between successive tokens of each instruction, and newlines separating instructions.

Output

The first and only line of the output should contain the shortest possible running time of the program, measured in processor cycles. If the program cannot stop, output should contain the word "HANGS".

Example

Sample Input	Sample Output
5 SET 0 1 JZ 4 0 RANDOM 0 JMP 1 STOP	6

6 Barcode

Barcode consistsof black and white vertical bars transformed to ones and zeroes by an optical reader. White and black bars alternate and can be thick or thin. A thin bar represents 0 and a thick bar represents 1, regardless of color. Thus, each barcode represents a sequence of digits.

Each bar in a barcode of a product appears as five "squares" high column. The width of a thin bar is one and the width of a thick bar is two "squares".

The reader used in this problem fell on the floor and since then it was unable to properly recognize the color of some "squares" of a barcode.

Write a program that from a given scanning of a bar code with our faulty reader will determine the binary sequence if at all possible.

Input

The first line of the input file contains an integer N, $1 \leq N \leq 100$, representing the total widthof the scannedbar code.

Eachof the next five lines contains N characters, each of which can be either "X", "."(dot), "?" (question mark). 'X' represents successfully recognized black "square", a dot represents successfully recognized white "square" and a question mark means that the reader couldn't determine the color of the "square".

Output

The only line of the output should contain a sequence of binary digits without separators that represent the barcode or the word "UNDETERMINABLE"(without quotes) if a unique binary sequence cannot be determined.

Example

Sample Input	Sample Output
4 .X?? .??. ??.? ?X.? .X?.	001

This page was intentionally left blank.

7 Bowlstack

Baking bread is my favourite spare-time pursuit. I have a number of stainless steel mixing bowls with straight sides, a circular bottom and a wider circular top opening. Geometrically, my bowls are truncated circular cones and for this problem, the thickness of the metal may be disregarded.

I store these bowls stacked in the natural way, that is with a common vertical axis, and I stack them in an order that minimises the total height of the stack. Finding this minimum is the purpose of your program.

Input

You should read all data from standard input.

On the first line of the input is a positive integer, telling the number of test cases to follow. Each case starts with one line containing an integer n, the number of bowls ($2 \leq n \leq 9$). The following n lines each contain three positive integers h, r, R, specifying the height, the bottom radius and the top radius of the bowl, and $r < R$ holds true. You may also assume that $h, r, R < 1000$.

Output

You should write all data to standard output.

For each test case, output one line containing the minimal stack height, truncated to an integer (note: truncated, not rounded).

Example

Sample Input	**Sample Output**
2 2 60 20 30 40 10 50 3 50 30 80 35 25 70 40 10 90	70 55

This page was intentionally left blank.

8 Street Directions

According to the Automobile Collision Monitor (ACM), most fatal traffic accidents occur on two-way streets. In order to reduce the number of fatalities caused by traffic accidents, the mayor wants to convert as many streets as possible into one-way streets. You have been hired to perform this conversion, so that from each intersection, it is possible for a motorist to drive to all the other intersections following some route.

You will be given a list of streets (all two-way) of the city. Each street connects two intersections, and does not go through an intersection. At most four streets meet at each intersection, and there is at most one street connecting any pair of intersections. It is possible for an intersection to be the end point of only one street. You may assume that it is possible for a motorist to drive from each destination to any other destination when every street is a two-way street.

Input

The input consists of a number of cases. The first line of each case contains two integers n and m. The number of intersections is $n (2 \leq n \leq 1000)$, and the number of streets is m. The next m lines contain the intersections incident to each of the m streets. The intersections are numbered from 1 to n, and each street is listed once. If the pair (i, j) is present, (j, i) will not be present. End of input is indicated by $n = m = 0$.

Output

For each case, print the case number (starting from 1) followed by a blank line. Next, print on separate lines each street as the pair "i j" to indicate that the street has been assigned the direction going from intersection i to intersection j. For a street that cannot be converted into a one-way street, print both "i j" and "j i" on two different lines. The list of streets can be printed in any order. Terminate each case with a line containing a single "#"(without quotes) character.

Note: There may be many possible direction assignments satisfying the requirements. Any such assignment is acceptable.

Example

Sample Input

```
7 10
1 2
1 3
2 4
3 4
4 5
4 6
5 7
6 7
2 5
3 6
0 0
```

Sample Output

```
1

1 2
2 4
3 1
3 6
4 3
5 2
5 4
6 4
6 7
7 5
#
```

9 Stairs

John is moving to the penthouse of a tall sky-scraper. He packed all his stuff in boxes and drove them to the entrance of the building on the ground floor. Unfortunately the elevator is out of order, so the boxes have to be moved up the stairs.

Luckily John has a lot of friends that want to help carrying his boxes up. They all walk the stairway at the same speed of 1 floor per minute, regardless of whether they carry a box or not. The stairway however is so narrowthat two persons can't pass each other on it. Therefore they deciced to do the following: someonewith a box in his hands is always moving up and someone empty-handed is always moving down. When two persons meet each other somewhere on the stairway, the lower one (with a box) hands it over to the higher one (without a box). (And then the lower onewalks down again and the higher one walks up.) The box exchange is instantaneous. When someone is back on the ground floor, he picks up a box and starts walking up. When someone is at the penthouse, he drops the box and walks down again.

After a while the persons are scattered across the stairway, some of them with boxes in their hands and some without. There are still a number of boxes on the ground floor and John is wondering how much more time it will take before all the boxes are up. Help him to find out!

Input

You should read all data from standard input.

One line with a positive number: the number of test cases. Then for each test case:

- One line with three numbers N, F, B with $1 \leq N, F \leq 1,000$ and $1 \leq B \leq 1,000,000$: the number of persons, the number of floors (0=ground floor, F=penthouse) and the number of boxes that are still on the ground floor.
- N lines with two numbers f_i and b_i with $0 \leq fi \leq F$ and $b_i = 0$ or $b_i = 1$: the floors where the persons are initially and whether or not they have a box in their hands (1=box, 0=no box).

Output

You should write all data to standard output.

One line per test case with the amount of time (in minutes) it will take to get all the remaining boxes to the penthouse.

Example

Sample Input	Sample Output
2 3 10 5 0 0 0 0 0 0 2 5 1 2 1 3 0	30 8