- Problems: There are 10 problems (20 pages in all, not counting this cover page) in this packet.
- Problem Input: Input to the problems are through the input files. Input filenames are given in the table below. Each input file may contain one or more test cases. Test cases may be separated by any delimiter as specified in the problem statements.
- Problem Output: All output should be directed to standard output (screen output).
- Time Limit: The judges will run each submitted program with certain time limit (given in the table below).

Table 1: Problem Information Sheet

	Problem Name	Input File	Time Limit
Problem A	Sudoku-Solver	pa.in	1 sec.
Problem B	Payment Efficiency	pb.in	1 secs.
Problem C	Computer Connectivity	pc.in	1 secs.
Problem D	Camera	pd.in	1 secs.
Problem E	Cubic Curve Fit	pe.in	2 secs.
Problem F	Hill Decipher	pf.in	2 secs.
Problem G	Treasure Island	pg.in	1 secs.
Problem H	Distinguishing Coloring of Trees	ph.in	1 secs.
Problem I	Baseball Team Elimination	pi.in	30 secs.
Problem J	Weight-constrained Maximum-Density Subtree	pj.in	3 secs.

Sudoku Solver

Input File: *pa.in*
Time Limit: *1 sec.*

Sudoku is a popular puzzle demanding logic and patience. The aim of the puzzle is to enter a numeral from 1 through 9 in each cell of a 9×9 grid made up of 3×3 subgrids, starting with various numerals given in some cells(called *given*). Each row, column and subgrid must contain only one instance of each numeral. See Figure 1 and Figure 2 for an example and its solution.

Write a program that output the correct solutions to the puzzles in the input file. You can solve these puzzles with or without the help of computer.

Input File Format

There is no need to read in the input file during your program execution as the input file is already provided to you. There are 20 Sudoku puzzles in the input file with the following format: Each row contains 9 consecutive numerals, 0 indicates that the cell has to be filled, and other numerals indicate that the cell is given that numeral. Each puzzle consists of 9 rows, there are 180 rows in total.

Output Format

Each row contains 9 consecutive numerals. There are 180 rows in total. And row 1 to 9 is the answer to the first puzzle and row $9i - 8$ to row $9i$ is the answer to the i^{th} puzzle.

For example, the puzzle in Figure 1 is encoded as the left column below, and its solution is encoded as right column.

```
530070000 534678912
600195000 672195348
098000060 198342567
800060003 859761423
400803001 426853791
700020006 713924856
060000280 961537284
000419005 287419635
000080079 345286179
```

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Figure 1: An example of a Sudoku puzzle

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Figure 2: The solution

Payment Efficiency

Input File: *pb.in*
Time Limit: *1 sec.*

Consider k kinds of coins with different values in dollars, and a toy of P dollars. The problem asks to calculate the minimum number of coins involved in the payment process for buying the toy. It is assumed that both parties, i.e., the buyer and the seller, have sufficient numbers of each kind of coin at their disposal. For example, suppose that there are 6 kinds of coins with values 1, 2, 5, 10, 20, 50 dollars, and the price of the toy is 83 dollars. If a father wants to buy the toy for his child, one possible payment process is that he pays for the toy with one 50-dollar coin, one 20-dollar coin, one 10-dollar coin and one 5-dollar coin, and receives one 2-dollar coin in change. That is, 50 + 20 + 10 + 5 - 2 = 83, which has 5 coins involved. Another possible payment process is: 50 + 20 +20 - 5 - 2. For this example, any possible payment process will have at least 5 coins involved, and hence 5 is the desired answer.

Input File Format

The first line of the input file contains the number of test cases. For each test case, there are integers on a line. The first integer is k ($1 \leq k \leq 10$), denoting the number of different kinds of coins. The second integer is P ($1 \leq P \leq 100$), denoting the price of the toy. There are k different positive integers in the rest of the line, denoting the values of the coins. Among the k integers, the smallest number is always 1, and the largest number is not greater than 100.

Output Format

For each test case, the output is a single line containing the minimum number of coins involved in the payment process for buying the toy.

Sample Input

```
3
6 83 1 2 5 10 20 50
6 36 1 24 34 39 46 50
6 42 1 2 3 7 19 72
```

Output for the Sample Input

```
5
3
4
```

Computer Connectivity

Input File: *pc.in*
Time Limit: *1 sec.*

Consider a set of N computers numbered from 1 to N, and a set S of M computer pairs, where each pair (i,j) in S indicates that computers i and j are connected. The *connectivity rule* says that if computers i and j are connected, and computers j and k are connected, then computers i and k are connected, too, no matter whether (i,k) or (k,i) is in S or not. Based on S and the connectivity rule, the set of N computers can be divided into a number of groups such that for any two computers, they are in the same group if and only if they are connected. Note that if a computer is not connected to any other one, itself forms a group. A group is said to be *largest* if the number of computers in it is maximum among all groups. The problem asks to count how many computers there are in a largest group.

Input File Format

The first line of the input file contains the number of test cases. For each test case, the first line consists of N $(1 \leq N \leq 30000)$ and M $(1 \leq M \leq 100000)$, where N is the number of computers and M is the number of computer pairs in S. Each of the following M lines consists of two integers i and j $(1 \leq i \leq N, 1 \leq j \leq N, i \neq j)$ indicating that (i,j) is in S. Note that there could be repetitions among the pairs in S.

Output Format

For each test case, the output should contain one number on a line, denoting the amount of computers in a largest group.

Sample Input

```
2
3 4
1 2
3 2
2 3
1 2
10 12
1 2
3 1
3 4
5 4
3 5
4 6
5 2
2 1
7 10
1 2
9 10
```

Output for the Sample Input

```
3
6
```

Camera

Input File: *pd.in*
Time Limit: *1 sec.*

AIP is a company that specializes in designing advanced digital cameras. In 2005, AIP has introduced a camera that can automatically classify the pixel in the image into four types: water, soil (e.g., crops), cement (e.g., highways) and others. For the 2006 model, some new features are to be added to the camera. A fundamental tool for the new features is the calculation of the largest single connected region of pixels of the same type. Furthermore, to make the camera marketable, this new feature must be done in real-time (less than 1 second). You are hired to add this functionality to the camera.

Technical Constraints

1. Image size is mxn, where $1 \leq m, n \leq 2,560$.
2. In the original image, water areas are denoted by pixels having value of 1, soil areas are denoted by pixels having value of 2, cement areas are denoted by pixels having value of 3, and the other areas are denoted by pixels having value of 0.
3. Two pixels are considered belonging to the same area if both of the following two conditions are met:
   (a) They both are of the same type (both are water, soil, cement, or others pixel type).
   (b) One pixel is connected to the other pixel in the north, south, east, or west direction.
   (c) Pixel connectivity is both symmetric and transitive.

Input File Format

The first line contains 1 integer indicating the number of test cases to follow. For each test case, the first line contains two integers, m, n, separated by a single space. Each of the next m lines contains n numbers, with each number being 0, 1, 2 or 3. The numbers on each line are connected without any space in between. There is a single line containing a single "." between two consecutive test cases.

Output Format

For each test case, output two numbers on a single line. The first number is the largest connected region type, {0, 1, 2, 3} in the image. The second number is the number of pixels in that region. You may assume that there is always a unique largest connected region in the test image.

Sample Input

```
3 4
3220
0111
1221
.
4 3
001
012
012
011
```

Output for the Sample Input

```
1 4
0 5
```

Cubic Curve Fit

Input File: *pe.in*
Time Limit: *2 seconds*

Given a set of N two-dimensional points $S = \{(x_i, y_i) \mid 1 \leq i \leq N\}$, the cubic curve fit problem is to find a cubic function $y = f(x) = ax^3 + bx^2 + cx + d$ such that the squared error $\epsilon = \sum_{i=1}^{N}(y_i - f(x_i))^2$ is minimized. This problem asks you to report four parameters a, b, c, d by *round-off* with the precision to 4 decimals for a given data set $S = \{(x_i, y_i) \mid 1 \leq i \leq N\}$.

Input File Format

The first line of the input file always contains one integer indicating the number of test cases to come. Each test data set consists of $N + 1$ lines, where the first line indicates the number of points N $(7 \leq N \leq 9)$ which is followed by N pairs of floating-point numbers to 2 decimals.

Input File Format

For each data case, report the estimated *parameters* a, b, c, d by round-off up to 4 decimals in a single line.

Sample Input

```
2
7
-3.01 -34.99
-1.99 -9.01
-1.01 1.01
0.01 0.99
0.99 -3.01
2.01 -5.01
2.99 1.01
8
2.00 -16.00
-1.00 0.00
-0.50 2.00
0.00 2.00
0.50 1.50
1.00 2.00
1.50 5.00
2.00 12.00
```

Output for the Sample Input

```
0.9916 -1.9713 -2.9593 0.9307
2.0000 -1.0000 -1.0000 2.0000
```

Hill Decipher

Input File: *pf.in*
Time Limit: *2 seconds*

Let Γ={A, B, C, $\cdots$, X, Y, Z } be the set of 26 English letters, and let the letters A, B, $\cdots$, Y, Z be represented as numbers 0, 1, $\cdots$, 24, 25, respectively. Denote $Z_{26} = \{0,\ 1,\ \cdots, 24, 25\}$. An integer linear transformation $(mod\ N)$ can be defined as $f\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = H \begin{bmatrix} x \\ y \end{bmatrix}$, where $a, b, c, d, x, y \in Z_N$. The inverse integer transformation exists only if $gcd(ad - bc, N) = 1$, that is, $(ad - bc)$ and N are relatively prime. This problem assumes that $N = 26$.

A Hill encipher takes a message, a character string consisting of English letters from Γ={A, B, C, $\cdots$, X, Y, Z }, as input and outputs an enciphered message of the same length based on applying an integer linear transformation successively on the pair of the characters of the input message string, for example, 'EUREKA' could be decomposed as EU,RE,KA, and is finally enciphered as 'KQXUMU' based on the transformation matrix $H = \begin{bmatrix} 9 & 13 \\ 2 & 3 \end{bmatrix}$. A corresponding Hill decipher takes 'KQXUMU' as input associated with the integer transformation matrix $H^{-1} = \begin{bmatrix} p & q \\ r & s \end{bmatrix} = \begin{bmatrix} 3 & 13 \\ 24 & 9 \end{bmatrix}$ which converts 'KQXUMU' back to 'EUREKA'. This problem asks you to design a *Hill decipher* based on a given Hill encipher to decipher an enciphered message.

Input File Format

The first line of the input file always contains one integer indicating the number of test cases to come. Each test data set consists of two lines, where the first line gives four integers a, b, c, d, corresponding to an integer transformation matrix $H = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ for the Hill encipher followed by an even integer n which is the length of the enciphered message. The enciphered character string of length n is given in the next line.

Output Format

Four integers p, q, r, s corresponding to the Hill decipher matrix H^{-1} are shown in the first line with the deciphered message appearing in the next line.

Sample Input

```
2
9 13 2 3 6
KQXUMU
5 2 13 5 8
MSMXIRCH
```

Output for the Sample Input

```
3 13 24 9
EUREKA
21 2 13 21
COMPUTER
```

Treasure Island

Input File: *pg.in*
Time Limit: *1 sec.*

A theme park called "Treasure Island" is newly opened. In the theme park, there are islands connected with bridges, where bridges are directed and one-way only. One major game of the theme park is a competition for collecting hidden treasures. A group of players may begin their search from an entrance island. In each island, some keys are placed in an obvious place. These keys can open a particular treasure if players can find it.
Image the keys and treasures have unlimited supply. Once a treasure is taken by some player, a new treasure will be supplied automatically. To prevent malicious players from collecting all the visible artifacts as their potential treasures, image a treasure is not removable if a player does not have correct keys at his/her hand to open the locks of treasures. In this theme park, players can choose different routes (even loops) to collect their treasures. Once they collect all the needed treasures, they can leave from an exit island.
You are the staff of the park. Everyday, you need to deploy keys and treasures. The deployment is different from day to day. In order to make the game easy and fair, a basic rule of key-treasure deployment is:

For any path (no matter how players choose their routes), a key should always appear first (shown to players) before its treasure's hidden island.

So, if a treasure X is hidden on some island Y, before a player visits island Y, a key for treasure X should be already shown to the player.

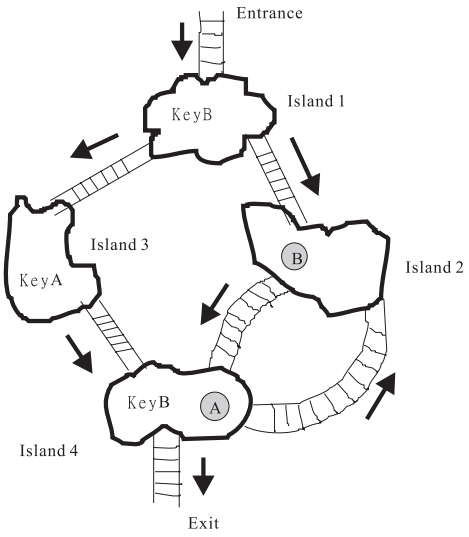

Figure 1: A sample deployment

Let a key of treasure X be *KeyX*. In figure 1, a deployment of treasures and keys is illustrated. In this deployment, treasure A is placed in island 4 and treasure B is placed in island 2. Three keys are placed in island 1,3, and 4 respectively.
In this sample deployment, treasure B satisfies the rule. Players choose any

not satisfy the rule – if a player chooses the route 1->2->4, he may see the treasure A without obtaining a $KeyA$ first.
Please write a program for the staffs of theme park. Given a deployment, please find out the treasures whose key-treasure deployments violate the basic rule.

Input File Format

The first line of the input data is a number N. N is the number of test cases. Each test case begins with numbers M and D. M is the number of islands. D is the number of deployment lines. By default, islands are indexed from 1 to M. Island 1 is the entrance island and island M is the exit island. Next, D lines of deployments are listed. Each deployment line begins with the island's index and then followed by keys or treasures. If a treasure X is deployed in the island, a string "X" is added in the deployment line, where X is "A"-"Z". If a $KeyX$ is placed in the island, a string "KX" appears in the deployment line. Items in a deployment line are separated by spaces. A "$" indicates there are no more items in the deployment line. NOTE: If an island is not listed in the deployment lines, it does not have treasures or keys deployed. NOTE: The number of directed bridges from an island to another island is not limited to one.

Following the deployment lines are information of directed bridges. Bridge information begins with a number B. B is the number of directed bridges. Next, B lines of bridges are followed. Each bridge is denoted by the index of source island and destination island.

Technical Specification

1. N $(1 \leq N \leq 20)$.
2. M $(1 \leq M \leq 1000)$.
3. B $(1 \leq B \leq 10000)$.

Output Format

For each test case, output the treasures (in capital letters) that violate the rule in one line with alphabetical order such as: ACDFG (without spaces)

Sample Input

```
1
4 4
1  KB  $
2  B  $
3  KA  $
4  A  KB  $
5
1 2
1 3
```

```
3 4
4 2
```

Output for the Sample Input

```
A
```

Distinguishing Coloring of Trees

Input File: *ph.in*
Time Limit: *1 sec.*

A rooted tree T consists of a set $V(T)$ of vertices (in this problem, the vertices are always represented by positive integers). One of the vertices, say $r \in V(T)$, is the *root* of the tree. Each vertex $v \in V(T)$ other than the root r has a unique *father*, where the root has no father. If v is the father of u, then we also say u is a *son* of v.

A 2-colored rooted tree is a rooted tree in which each vertex is colored by one of the two colors: black and white.

Figure 1 below shows a 2-colored rooted tree, where each white vertex is represented by an unfilled circle, each black vertex is represented by a filled circle. The father-son relation is represented by an arrow: if there is an arrow from vertex u to vertex v then v is the father of u.

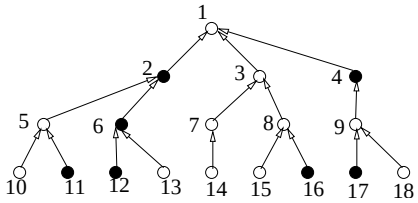

Figure 1: A 2-colored rooted tree T.

An automorphism of T is a permutation σ of its vertex set $V(T)$ (i.e., a one-to-one mapping from $V(T)$ to $V(T)$) such that if v is the father of u, then $\sigma(v)$ is the father of $\sigma(u)$. Moreover, for any vertex v, $\sigma(v)$ is black if and only if v is black. Note that for any rooted tree T, the identity map, id, defined as $id(x) = x$ for all vertices x is an automorphism. This automorphism is called the trivial automorphism.

A 2-coloring of a rooted tree is called distinguishing if it only has the trivial automorphism. Two 2-colorings of T, say f and g, are called isomorphic if there is a permutation σ of the vertices of T such that if v is the father of u, then $\sigma(v)$ is the father of $\sigma(u)$. Moreover, for any vertex v, $\sigma(v)$ is black under the coloring f if and only if v is black under the coloring g.

Given a rooted tree T, your task is to compute the number of non-isomorphic distinguishing 2-colorings of T.

Input File Format

The input consists of a number of rooted trees, say $T_1, T_2, \cdots, T_m$, where $m \leq 20$. Each tree has at most 100 vertices. Information of each of the rooted trees are contained in a number of line. Each line starts with a vertex (which is a positive integer) followed by all its sons (which are also positive integers), then followed by a 0. Note that 0 is not a vertex, and it indicates the end of that line. Consecutive integers in a line are separated by a single space. A line containing a single 0 means the end of that tree. The next tree starts in the next line. Two consecutive lines of single 0 means the end of the input.

Output Format

which is the number of non-isomorphic distinguishing 2-colorings of T.

Sample Input

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

Output for the Sample Input

```
2
0
2
8
```

Sample Input Explanation

The input consists of four trees. The first tree T_1 has root 1, and 1 has two sons: 2 and 3. This tree has two distinguishing 2-colorings: Any distinguishing 2-coloring must color exactly one of $2, 3$ by black color. The vertex 1 can be colored either black or white. Observe that the coloring which colors $1, 2$ black and color 3 white is isomorphic to the coloring which colors $1, 3$ black and colors 2 white.

The second tree has root 1, which has three sons: $2, 3, 4$. This rooted tree has no distinguishing 2-colorings. For any 2-coloring of this tree, two vertices of $2, 3, 4$ receive the same color. Without loss of generality, assume that $2, 3$ receive the same color. Then the permutation interchanges 2 and 3, and fixes every other vertex is a non-trivial automorphism of this 2-colored tree.

Basketball Team Elimination

Input File: *pi.in*
Time Limit: *30 sec.*

Dr. Jay was a famous basketball player and now he is still crazy about basketball games. However, he only has interest in the teams with a chance to win the championship. Consequently, he loves watching the games played by those teams and hates spending time on the losing teams. A team is eliminated at some point during the season, if it cannot finish the season as the champion.

There are numerous basketball leagues all over the world. Dr. Jay is tired of figuring out himself which teams still have the chance to become the champion for each league. Dr. Jay asks you to write a program for him to find the eliminated teams of a league automatically.

Suppose all the teams can only win or lose a game, i.e., there is no tie. At the end of the season, the team winning most games is the champion. If there is a tie, then there will be extra games to determine the final championship. Now, write a program to compute which teams are eliminated based on current winning records and the remaining games.

Input File Format

The input file begins with the number of leagues M, i.e., the number of test cases. For each league, it starts with a number N. The following N lines are the names of the N teams. The next N lines are the current records of the corresponding teams, where each of them contains N +1 numbers. The first number indicates how many games the corresponding team has won, and the I-th of the following N numbers is the number of remaining games against the I-th team. The input of a league ends with a line with "@" alone. The number of teams is at most 100 and the number of remaining games between any two teams is no more than 10.

Output Format

For each league, the output should consist of the names of eliminated teams which are listed in the order as appearing in the input file. If there is no team eliminated, then print "No team is eliminated!"

Sample Input

```
2
6
Suns
Bulls
Jazz
Spurs
76ers
Heat
60 0 1 2 1 2 1
70 1 0 2 2 1 1
71 2 2 0 1 1 2
```

```
53 2 1 1 1 0 2
42 1 1 2 1 2 0
@
2
Suns
Spurs
2 0 5
0 5 0
```

Output for the Sample Input

```
League #1:
Suns
Spurs
76ers
Heat
League #2:
No team is eliminated!
```

Weight-constrained Maximum-Density Subtree

Input File: *pj.in*
Time Limit: *3 sec.*

Given a sequence S of n integer pairs $[a_i, w_i]$ with $w_i > 0$ for $i = 1, 2 \ldots, n$, and two integer weight bounds $w_{\min}$ and $w_{\max}$, the *weight-constrained maximum-density segment problem* is to find a consecutive subsequence $S(i, j)$ of S such that the density of $S(i, j)$, i.e., $\lfloor \frac{a_i + a_{i+1} + \ldots + a_j}{w_i + w_{i+1} + \ldots + w_j} \rfloor$ is maximum over all other subsequence of S satisfying $w_{min} \leq w_i + w_{i+1} + \ldots + w_j \leq w_{max}$. This problem arises from the investigation of non-uniformity of nucleotide composition within genomic sequences, which was first revealed through thermal melting and gradient centrifugation experiments. Researchers observed that the compositional heterogeneity is highly correlated to the GC content of the genomic sequences, and this motivates finding GC-rich segments.

Here you need consider a related problem on trees. Before proceeding to describe the problem, some necessary definitions in terms of graph theory are first given. A graph G is a pair (V, E), where V is a finite set and E is a subset of $\{(a, b) \mid (a, b)$ is an unordered pair of $V\}$. For an edge $e = (u, v)$, u and v are end-vertices of e, and both vertices are *adjacent* each other. A *path* $P = \langle v_0, v_1, \ldots, v_t \rangle$, is a sequence of distinct vertices such that any two consecutive vertices are adjacent. A graph G is *connected* if every pair of vertices is connected by a path in G. A *subgraph* of $G = (V, E)$ is a graph (V', E') such that $V' \subseteq V$ and $E' \subseteq E$. A *tree* is a connected graph and has no cycles. For a tree T, a *subtree* of T is a connected subgraph (V', E') of T, where $V' \subseteq V$ and $E' \subseteq E$. Note that a single vertex can also be regarded as a subtree.

The problem is now described as follows. A biologist, William, attempts to analyze the ancestor-descendant relation for n species connected by a tree $T = (V, E)$, called an *evolutionary tree*. Each species v of T is associated with a *value-weight pair* $[val_v, w_v]$, where *value* val_v is an integer and *weight* w_v is a non-negative integer. The *density* of T, denoted by $d(T)$, is defined as $\lfloor \frac{\Sigma_{v \in V} val_v}{\Sigma_{v \in V} w_v} \rfloor$. The *weight-sum of* T, denoted by $w(T)$, is $\Sigma_{v \in V} w_v$. William believes that those species form a maximum-density subtree in T (satisfying a given weight constraint) diverge from a common ancestor. He formulates the following problem: Given an evolutionary tree $T = (V, E)$ of n specifies associated with value-weight pairs, and two bounded integers $w_{\min}$ and $w_{\max}$, compute the density of a maximum-density subtree T' in T such that $w_{\min} \leq w(T') \leq w_{\max}$.

Assume that T has $2 \leq |V| \leq 2000$ specifies represented by $\{1, 2, \ldots, |V|\}$, $1 \leq w_{\min} \leq 10000$ and $1 \leq w_{\max} \leq 10000$. Please write a program to compute the density of a weight-constrained maximum-density subtree of T. If such a tree does not exist, your program needs to output -1. For example, given $w_{\min} = 4$ and $w_{max} = 100$, the density of a weight-constrained maximum-density subtree $T' = (V', E')$ of a tree $T = (V, E)$ shown in Figure 1 equals 4. Note that the vertex $V' = \{4, 5, 6, 7\}$ and $E' = \{(4, 5), (4, 6), (6, 7)\}$.

Input File Format

The input consists of a number of test cases. Each test case consists of one tree, which has the following format: The first line contains two numbers,

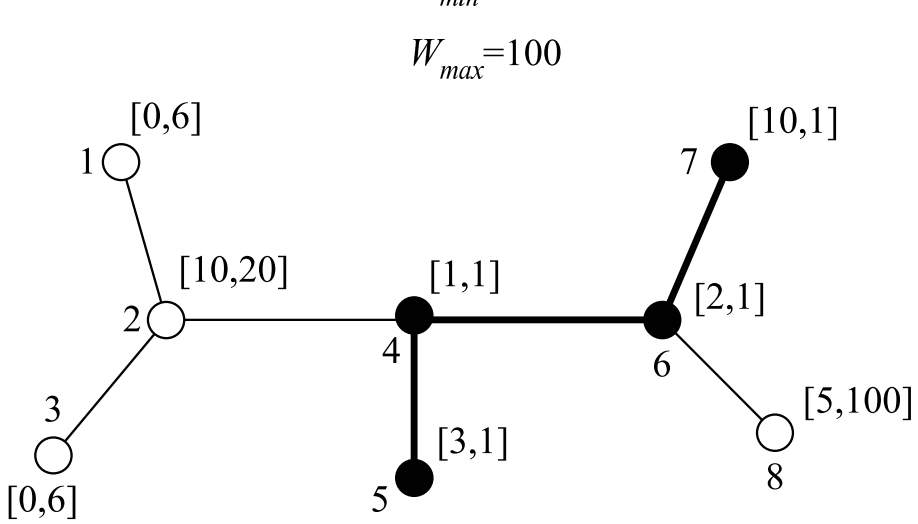

Figure 1: An illustration of a weight-constrained maximum-density subtree in a tree

w_{min} and w_{max}, separated by a single space. The second line contains one positive integer $n \leq 2000$, which is the number of species of the input tree. The next n lines contain n species such that one line contains one specifies and the corresponding value and weight. Each line is represented by three integers separated by a single space; the first number represents a species, the second one represents its value, and the third one represents its weight.
The next line contains one positive integer $m = n - 1$, which is the number of edges of the input tree. The next m lines contain m edges such that one line contains one edge. Each edge is represented by two positive integers separated by a single space; the first integer represents one end-vertex of the edge, and the second one represents the other end-vertex. Finally, a 0 at the $(n+m+4)$th line indicates the end of the first test case.
The next test case starts after the previous ending symbol 0. A -1 singles the end of the whole inputs.

Output Format

The output contains one line for each test case. Each line contains an integer, which is the density of a weight-constrained maximum-density subtree of the corresponding input tree.

Sample Input

```
4 100
8
1 0 6
2 10 20
3 0 6
4 1 1
5 3 1
6 2 1
```

```
8 5 100
7
1 2
2 3
2 4
4 5
4 6
6 7
6 8
0
14 16
6
1 0 100
2 0 100
3 6 6
4 8 8
5 0 100
6 0 100
5
1 3
2 3
3 4
4 5
4 6
0
-1
```

Output for the Sample Input

```
4
1
```