Problem A: I Know a Guy, Who Knows a Guy ...

The head of the East End Mafia, Don Craig, has decided he would like to get Happy 81st Birthday text-messages from all m, $1 \leq m \leq 26$, city councillors that owe money to his organization. Each of his $n \geq 1$ Wiseguys talks to a subset of the $l \geq 1$ lower-level mobsters, each of whom in turn talks to particular subsets of the mobsters and the councillors. It takes 5 minutes for either a Wiseguy or a mobster to contact everyone he knows with a request, and a further 5 minutes for a contacted councillor to text-a message the Don. The Don has decided the Wiseguy who can get all of the councillors to text-message the Don in the shortest amount of time shall be the new Capo.

Write a program which, given a description of the mobsters that can be contacted by each Wiseguy and the mobsters and clients that can be contacted by each mobster, determines the Wiseguy who gets to be the new Capo, as well as the time it took for all clients he contacted through his network to text-message the Don. Note that contacts are not symmetric, *e.g.*, mobster A can contact mobster B but B cannot contact A, a mobster may be contactable by more than one Wiseguy, a client may be contactable by more than one mobster, and no Wiseguy contacts another Wiseguy.

Input: Each Wiseguy has a single-word Family nickname, each mobster has a unique positive non-zero Family ID number in the range 1 to l, and each councillor has a unique Family ID name consisting of a single lower-case letter. The input will consist of a $(m + l + 1)$-line file, where line 1 specifies n, l, and m, lines 2 to $(n + 1)$ specify the mobster-contacts of each Wiseguy (one line per Wiseguy), and lines $(n + 2)$ to $(n + l + 1)$ specify the mobster- and councillor-contacts of each mobster (one line per mobster). Each Wiseguy's contact line specifies the Wiseguy's name, the number of mobsters he can contact, and the Family ID numbers of those mobsters; similarly, each mobster's contact line specifies the mobster's Family ID name, *i.e.*, **CT**x where x is the mobster's Family ID number, the numbers of other mobsters and councillors that he can contact, and the Family ID numbers and names of these mobsters and councillors. Note that all mobsters contact-lines are given in ascending order by mobster Family ID number.

You may assume that all input files are formatted correctly, and that there is always exactly one Wiseguy with minimum all-councillor contact-time.

Sample Input (available as file A.in):

```
2 3 4
Guido  2 1   2
Eddie  2 1   3
CT1    2 2   1 3   a b
CT2    1 3   3     b c d
CT3    2 2   2 3   b c
```

Sample Output (available as file A.out):

```
New Capo: Guido [total contact time = 10 min]
```

Problem B: What Makes You Happy Makes Us Happy

You run a small photocopying service with a single large machine. Every morning you get a set of jobs from customers. You want to do the jobs in the order that keeps the customers happiest. Let customer i's job takes time t_i to do. Given a schedule, an ordering of all the jobs, let c_i be the time at which this job is completed, which equals the sum of all the times of all the jobs that are done before plus t_i. For example if job i is done first then $c_i = t_i$, and if job i is done immediately after job j, then $c_i = c_j + t_i$.

The happiness of a customer is assumed to be proportional to the completion time of their job. You want to minimize the sums of the completion times, $c_1 + c_2 + \ldots c_n$. You need a program that, when the times of the customers jobs are given, returns the best order in which to do the jobs.

Input:
The first line of input contains the number n of jobs for a single day's work. The next n lines contains the name of a customer as a character string without blanks, followed by whitespace, and then an integer that is the length of time it takes to do their job. The names of customers will always be different, and will always start with a letter of the alphabet. There can be multiple cases in the file, so that if there is more data, it will start with another integer n. Data is finished when a blank line is encountered.

Output:
For each case, output the names of the customers in the order required to minimize the sum of the completion times, with one name per line. If there is more than one possible ordering that produces the minimum sum, then the correct order is the one that is as close as possible to the order of the jobs in the input (since someone who submitted a job first would prefer to get it back first). The last line for each case should be the sum of the completion times. Leave a blank line between each case.

Sample Input (available as file B.in):

```
2
A 1
B 2
3
A 3
B 2
cc 1
```

Sample Output (available as file B.out):

```
A
B
4

cc
B
A
10
```

Problem C: Kakuro

Kakuro is a form of logic puzzle related to Sudoku. It is played on a rectangular grid. Some cells are occupied with a '\' . A sequence of consecutive empty cells in a row or column is called a "run". To solve a puzzle you must put a digit 1 to 9 in each empty cell and the digits in a run must be different. The sum of the digits in each run is given to help solve the puzzle. For a row run the total is placed after the '\' just to the left of the run. For a column run the total is placed before the '\' just above the run. The largest possible total for a run is 45 (=1+2+ ...+9). Every empty cell is part of both a row run and a column run and every puzzle has a unique solution. Consider the following sample puzzle:

```
\   |17\   |24\   |   \   | 3\   | 9\
\16|       |       |24\4  |       |
\24|       |       |       |16\8  |
\   | 3\24|       |       |       |   \
\3  |       |   \17|       |       |   \
```

This 5 by 6 puzzle has 7 row runs and 7 column runs. The notation $r(t, n)$ denotes a run with n cells whose entries must total t. For example, $r(17, 2)$ describes the run in column 2, rows 2:3. Note that the only 2 distinct digits which add to 17 are $\{8, 9\}$. Similarly $r(16, 2)$ in row 2 must contain 7,9 and the four occurrences of $r(24, 3)$ must contain $\{7, 8, 9\}$. (Any other combination of 3 distinct digits would have a smaller total.) The $r(9, 2)$ in the last column has four possible decompositions. Runs with unique decompositions make Kakuro puzzles easier to solve.

Let $U[i]$ denote the set $\{1, 2, ..., i\}$ and let $U = U[9]$. There are 512 possible subsets of U which may be used to fill runs. A subset S is "special" if no other subset of U has the same sum and same number of elements, denoted $|S|$. These special subsets occur often in solving Kakuro puzzles. If S is special then it is the only candidate for the entries in a run of the form $r(sum(S), |S|)$. The following subsets are the only ones which are special:

- any singleton set: $\{1\}, \{2\}, \ldots, \{9\}$;
- the sets $U[i]$ (since any other subset of the same size has a larger sum);
- the sets $U[i] + \{i + 2\}$, i.e. $\{1, 3\}, \{1, 2, 4\}, \ldots, \{1, 2, 3, 4, 5, 6, 7, 9\}$;
- if S is special so is $U - S$ (the complement of S).

A run $r(t, n)$ is special if there is a special set S such that $t = sum(S)$ and $n = |S|$. We associate this set S with the run. For this problem we associate the set U with non special runs. Each empty cell is part of a row run and a column run each of which has an associated set.

To solve (in part) a Kakuro puzzle use the following rules repeatedly as long as they apply.

- If there is only one number common to both the row and column sets of a cell it must be the value for the cell and this value is removed from both sets.
- If a run has only one (remaining) empty cell then that cell's value can be calculated.

Using these rules the sample Kakuro above can be solved completely and the output for it is the first 5 lines of the sample output below.

Input and Output:
The input consists of one or more puzzles separated by empty lines. Each puzzle grid will have 2 to 20 rows or columns. An r by c puzzle will appear as r lines of $6c$ characters (including the newline). Note that on Windows a newline '\n' may have an extra '\r' character, in which case we might have r lines of $6c + 1$ characters. The output will be the same as the input except that some of the empty cells will have a digit '1' to '9' in the middle.

Sample Input (10 lines, available as file C.in):

```
\   |17\   |24\   |   \   | 3\   | 9\
\16|       |       |24\4 |       |
\24|       |       |       |16\8 |
\   | 3\24|       |       |       |   \
\3 |       |   \17|       |       |   \

\   |19\   |20\   | 6\
\20|       |       |
\18|       |       |
\6 |       |       |
```

Note that the rules given are not enough to solve the second puzzle.
Sample Output: (available as file C.out):

```
\   |17\   |24\   |   \   | 3\   | 9\
\16|   9   |   7   |24\4 |   3   |   1
\24|   8   |   9   |   7   |16\8 |   8
\   | 3\24|   8   |   9   |   7   |   \
\3 |   3   |   \17|   8   |   9   |   \

\   |19\   |20\   | 6\
\20|       |       |
\18|       |       |
\6 |       |       |
```

Problem D: Personalized Especially for You

Often you want to do some task which involves taking some "template" and making a few substitutions to create a finished "product". For example, you might want to take a sample letter and individualize it by putting in the specific address, name, phone number, and so on. This letter can then be printed and mailed or e-mailed directly.

To create these letters, you need to write a program which takes a mail message template and a collection of "records", where each record has an e-mail address, a subject, and a number of substitutions. For each of the records, your program will create a mail message body by making each substitution on the original template, and then e-mail the message to the respective recipient with the respective subject line.

The input begins with a template, which ends with a line containing only "|| " (as a separator – this line should not appear in the letters). It is followed by an unknown number of records, ending at the end of file. Each record has the following format:

```
recipient e-mail|subject
match-pattern1|replacement-text1|
match-pattern2|replacement-text2|
...
match-patternN|replacement-textN|
```

You may assume that '|' is not used anywhere but as a separator. The match-patterns do not have newline characters in them, but the replacement texts can. The records are separated by blank lines; note that since the replacement text can be multi-line, it could also contain a blank line. Different records do not necessarily have the same number of substitutions.

Sample Input (available as D.in):

```
<address>
<letter date>

Dear #name:

Please send me all your money by NEEDED-DATE.

                    Sincerely,
                    you know who.
||
sholmes@doyle.org.invalid|ATTENTION
<address>|221b Baker Street
London
England|
<letter-date>|August 17, 1898|
#name|Sherlock|
NEEDED-DATE|August 24, 1898|
```

jwatson@doyle.org.invalid|Immediate business opportunity
<address>|221b Baker Street
London
England|
<letter-date>|August 18, 1898|
#name|John|
NEEDED-DATE|August 29, 1898|

You can assume that the input is correctly formatted.

Since actually e-mailing messages is not possible in this situation, your program just outputs the text of the letter, with each letter followed by a line containing only "-- ".

Sample Output (available as D.out):

Recipient: sholmes@doyle.org.invalid
Subject: ATTENTION
Message body:
221b Baker Street
London
England
August 17, 1898

Dear Sherlock:

Please send me all your money by August 24, 1898.

Sincerely,
you know who.
--
Recipient: jwatson@doyle.org.invalid
Subject: Immediate business opportunity
Message body:
221b Baker Street
London
England
August 18, 1898

Dear John:

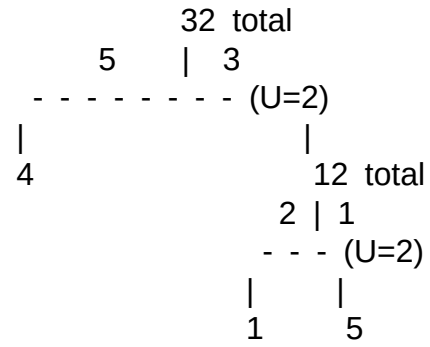
Please send me all your money by August 29, 1898.

Sincerely,
you know who.
--

Problem E: Seeking Equilibrium

A decoration factory makes a decorative structure that can be attached to the ceiling of a room. The problem is to hang these structures so that they are balanced. The simplest structure consists of a simple weight. More elaborat structures consist of a cylindrical rod with simpler structures attached at each end with short very thin wires. This forms a hierarchy. Where smaller structures attach to the end of a rod they are free to rotate to avoid collisions with other parts of the structure hierarchy. One wire attaches to the ceiling. The others attach at the top to the end of a rod and at the bottom to a weight or to another rod at a "pivot point" somewhere between the ends so that the rod is balanced and hangs horizontal. Your program is to calculate the pivot points.

The following figure shows an example of two rods and 3 suspended weights.



The input describing this structure would be:
(L: 8 U: 2 (W: 4) (L: 3 U: 2 (W: 1) (W: 5)) Here L indicates the length of a rod whose weight per unit length is U and (W: n) represents a weight of size n.

The corresponding output would be:
(L: 8.0 U: 2.0 P: 5.0 T: 32.0 (W: 4.0) (L: 3.0 U: 2.0 P: 2.0 T: 12.0 (W: 1.0) (W: 5.0)))
Here P indicates the pivot point of a rod and T its total weight including that of the substructures at each end. So the first rod of length 8 is suspended at a point 5 units from the left end (and 3 from the right). The second rod is balanced when the pivot point is 2 units from the left end (and 1 from the right). The total weight of the second rod is 12 ($3 \cdot U + 1 + 5$). The total weight of the whole hierarchy is 32.

All weights and lengths are given using the same weight unit and the same measuring unit respectively. The wires are assumed to have 0 weight.

Hints:

1. A rod is balanced if the bending moment on each side is the same
2. The moment generated by a weight w situated at P distance from the pivot is equal to $w \cdot P$
3. The moment generated by the weight of a rod of a length L and attached at a distance P from its end is equal to
    - Left side: the weight of the section P of the rod $P \cdot U$ multiplied by $\frac{P}{2}$

- Right side: the weight of the section $(L - P) \cdot U$ multiplied by $\frac{(L-P)}{2}$

Input and Output: The input consists of multiple cases each describing a structure on one line. A structure is either:

- an expression (W: n) where n is a number in which case the output is the same as the input except that n is printed to one decimal place (and the total for this is n);
- an expression (L: n1 U: n2 ¡x¿ ¡y¿) where ¡x¿ and ¡y¿ are smaller structures and n1 and n2 are numbers. In this case the output is (L: n1 U: n2 P: n3 T: n4 ¡Ox¿ ¡Oy¿) where the numbers are printed to one decimal place and ¡Ox¿ and ¡Oy¿ are the output for ¡x¿ and ¡y¿, respectively, and P gives the pivot point and T the total weight of the rod and the sub-structures at each end.

All numbers are to be output to one decimal place.

Sample Input (available as E.in):

```
(W: 4)
(L: 3 U: 2 (W: 1) (W: 5))
(L: 8 U: 2 (W: 4) (L: 3 U: 2 (W: 1) (W: 5)))
(L: 12 U: 0.5 (L: 3 U: 2 (W: 1) (W: 5)) (L: 3 U: 2 (W: 1) (W: 5)))
```

Sample Output (available as E.out):

```
(W: 4.0)
(L: 3.0 U:2.0 P: 2.0 T: 12.0 (W: 1) (W: 5))
(L: 9.0 U:2.0 P: 5.0 T: 32.0 (W: 4.0) (L: 3.0 U:2.0 P: 2.0 T: 12.0 (W: 1) (W: 5)))
(L: 12.0 U: 0.5 P: 6.0 T: 30.0 (L: 3.0 U:2.0 P: 2.0 T: 12.0 (W: 1) (W: 5)) (L: 3.0 U:2.0 P: 2.0 T:
```

Problem F: Over, Under, Through

You are in a huge city laid out as a grid, and all places are represented by a pair of integer coordinates (x, y). You are the dispatcher for a local courier company; unfortunately your company does not own any vehicles yet. For each package you have four choices, taxi, subway, or helicopter. Each method of transport has its own constraints. Taxis can only travel using the roads in the grid, so their distance travelled (the *grid_distance*) is the sum of the x-coordinate distance and the y-coordinate distance. Helicopters instead can travel in a single straight line from one point to the other (so the distance is the *euclidean_distance*). Subways, however, travel in the same ways as taxis but are charged in concentric zones around $(0, 0)$; the cost for a subway trip is a linear function of how many zones you need to travel through. The zone of a point is equal to one-fifth of the point's distance from $(0, 0)$, rounded up. If we want to travel between a point in zone a and a point in zone b, with $a < b$, we would need to travel through $1 + b - a$ zones.

You have an infinite supply of employees to do the deliveries. You want to find the cheapest way based on the following costs:

taxi: time is $a \cdot (grid\_distance)$ minutes
cost is $b \cdot (grid\_distance)$

subway: time is $c \cdot (grid\_distance)$ minutes, where $c > a$
cost is $d \cdot (number\_of\_zones)$

helicopter: time: $e \cdot (euclidean\_distance)$ minutes
cost $f \cdot (euclidean\_distance) + g$

While some transportation costs more, you cannot always take the cheapest route; if you take longer than the deadline specified by the contract, you must pay a specified penalty. This penalty needs to be added to the cost of travel in order to find out the true cost of the transport option.

The input file consists of one initial line, specifying the values of a, b, c, d, e, f, and g (in that order). Each subsequent line of input (ending at the end of file) gives information about a specific delivery contract. Each of these lines gives:

o_x o_y d_x d_y *start_time* *deadline* *penalty_per_minute*

The output should be, for each case, the cheapest transport option and its cost, with the cost expressed to two decimal points.

Sample Input (available as F.in):

```
2 3 5 1 1.5 5 15
-1 -7 2 2 32 60 1
-3 4 1 5 45 52 2
```

Sample Output (available as F.out):

```
subway: 34
helicopter: 35.62
```

Problem G: Signal to Noise

Albert (a spy) and Bill (his contact) are trying to communicate over a long distance, and have a telegraph connection available to them. Unfortunately this connection has been preprogrammed with two signals, each of which will repeat indefinitely once it has been switched on. Albert has been investigating a particular matter, and needs to send a yes/no response to Bill. They have agreed which of the signals means "yes" and which means "no", but they are having some difficulty telling the two signals apart, especially since each signal repeats continuously. Bill wants to find a short string that he can listen for; as his newest minion, this is your task.

Given the two signals, each expressed as a sequence of dots (.) and dashes (-), you need to determine the shortest substring (consecutive symbols) that appears in the repetition of the first string and does not appear in the repetition of the second string. If there are many equally shortest strings, any one of them will do.

The first line of the input file is a single number n that indicates how many cases there are. It is followed by n lines, each containing two strings, separated by a space. Your output should be n lines, each giving a shortest string that is a substring of the first input string for that case and is not a substring of the second input string for that case.

Sample Input (available as G.in):

```
2
-.-.-. ..-
....---  .....---
```

Sample Output (available as G.out):

```
-.-
-....-
```