

ACM International Collegiate Programming Contest Asia Regional Contest, Singapore 2000

Contest Problem Sets

*Nanyang Auditorium
Nanyang Technological University
Singapore
6-7 December 2000*

Contents

Problem 1	2
Problem 2	4
Problem 3	8
Problem 4	10
Problem 5	11
Problem 6	11
Problem 7	17
Problem 8	20

Problem 1

Counting Triangles

Input file: data1.txt

This is a game of counting triangles. Short sticks of 1 or $\sqrt{2}$ units of length are placed onto a grid as shown in Figure 1, horizontally, vertically or diagonally. The sticks placed diagonally are allowed to cross each other.

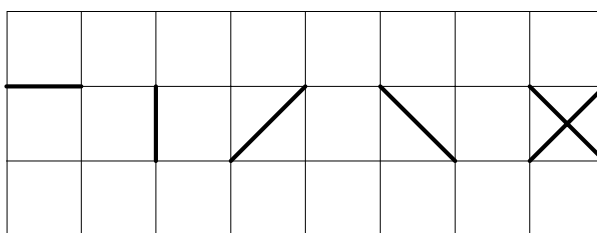


Figure 1: Basic placement patterns of sticks.

A random placement of short sticks on a grid creates either a pattern without any triangle or a pattern with one or more triangles as illustrated in Figure 2 where patterns (a), (b), (c), (d) and (e) have 2, 5, 12, 0 and 0 triangles respectively.

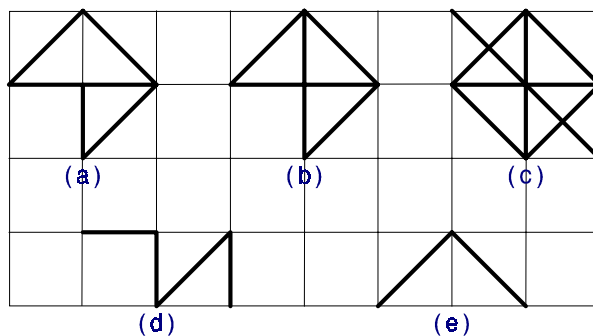


Figure 2: Sample patterns of triangles.

Your task is to write a program that counts the number of triangles in a pattern.

Input

The input file consists of one or more patterns of short sticks. The first line of each pattern contains the name of the pattern. Each name is an alphanumeric string of no more than 30 characters. The next line contains the number of short sticks to be placed on the grid. This is followed by pairs of integers specifying the starting and ending coordinates of each stick. The grid will not be larger than 10 by 10; i.e., the coordinates of the lower left and upper right corners

are (0, 0) and (9, 9) respectively.

Output

For each pattern, the output file contains a line with the name of the pattern followed by the number of triangles found. Sample inputs and outputs for Figures 2(a) and 2(e) are illustrated below.

Sample Input

Figure2(a)

6

3 2 2 2 2 1 2 3 2 2 3

2 3 1 2 2 2 2 1 2 1 3 2

Figure2(e)

2

1 1 2 2 2 2 3 1

Sample Output

Figure2(a) 2 triangles

Figure2(e) 0 triangles

Problem 2

Treasure Hunt in 3-Dimensional Maze

Input file: data2.txt

A treasure hunt takes place in a 3-dimensional space filled with rocks and electric monsters. The space is of at most a $10 \times 10 \times 10$ cubic unit block. Your spaceship is not of negligible size; it has to follow the instructions given below for navigation. Suppose you are at location marked “*” (Figure 1), you can move, in one step and one unit of time, to the center of one of the following positions:

1. e, k, m, n, p or v if this position is empty (or is the destination).
2. $b, d, f, h, j, l, o, q, s, u, w$ or y if this position is empty (or is the destination), and the 2 common nearest neighbors between * and this position are empty too. For example, one can go to q if p, q , and n are empty.
3. a, c, g, i, r, t, x or z if this position is empty (or is the destination), and the 6 common neighbors between * and this position are empty too. For example, one can go to z if n, p, q, v, w, y and z are empty.

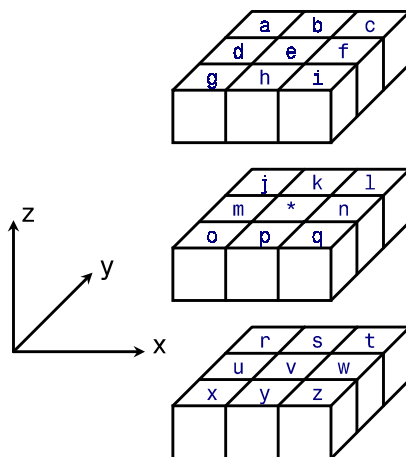


Figure 1: A 3-dimensional point and its neighbors shown in 3 layers of cubic blocks.

The treasure is placed in the destination block. The homes of electric monsters are randomly located. Each home has exactly one monster and is located at the center of a block. Each monster will attack objects that it can see in an empty block. An empty block is visible if all its eight corners and center can be seen; i.e., it is not blocked by rocks or the destination block. The size of a rock is 1 cubic unit and the destination block is safe from monster attack. Each attack cycle takes one unit of time and aims at the center of an empty block where your spaceship is located. For example, consider the top layer in Figure 1. If all the locations are empty except a (a cubic rock) and d (a monster in the block center), the monster will be able to see c, e, f, g, h and i but not b . The power of an electric blow from each monster follows an exponential function as follows:

$$power = 2e^{-d^2/\sigma^2}$$

where d is the distance and σ is the monster size. The distance d is computed from the center of one block to the center of the other. When the accumulated electric exposure of a trip exceeds or equals to 1, serious damage will be done to your spaceship and you are out of the game. Hence, one must determine if there is a safe pass from the starting point to the ending point where the treasure is located. In order to minimize the damage done to the spaceship, one must compute the safest pass that minimizes the total exposure of electric attacks. When there are more than one safest passes, one should output the pass with the shortest time spanned. If there are more than one such passes, printing out one pass is good enough.

You should note that:

- Time efficiency is an important consideration in this problem.
- To handle the problem of truncation error in real number representation, the accuracy of computation will only require 4 digits after the decimal point.

Input

The input file consists of one or more maze patterns as illustrated in Figure 2 (and in the Sample Input below). The first line of each pattern contains the name of the maze. Each name is an alphanumeric string of no more than 30 characters. The next line contains the dimension specification in the order of X, Y and Z. The coordinate origin starts from (0,0,0). The input continues with one or more layers of specifications of content in each cubic volume where -2 represents the destination, -1 represents a rock, 0 represents an empty space, and any non-zero positive integer represents the size of a monster present in that location. The specification starts from the top layer to the bottom layer. At each layer, it starts from the top row to the bottom row as shown in the Sample Input. Each pattern ends with the starting and ending positions of the hunt.

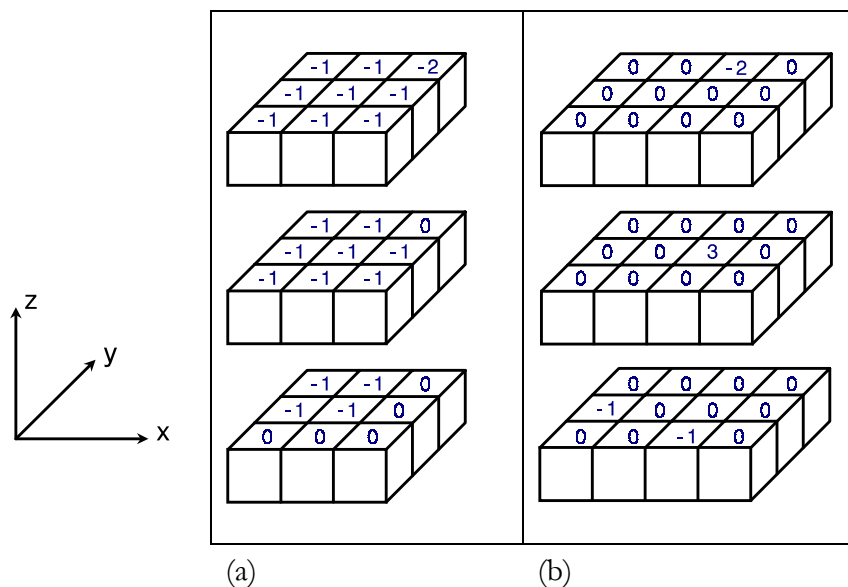


Figure 2: Two examples of input mazes.

Output

For each maze pattern, the output should contain a line with the name of the maze followed by

the accumulated electric exposure of the safe path. Four precision digits after the decimal point are required; i.e., the other digits after the decimal point should be truncated. If there is no safe path, a negative number should be printed. Otherwise, the safe path coordinates should be printed as shown in the Sample Output. Each line should start with a <tab> and is followed by exactly 4 coordinate points.

Sample Input

Figure2(a)

3 3 3

-1 -1 -2

-1 -1 -1

-1 -1 -1

-1 -1 0

-1 -1 -1

-1 -1 -1

-1 -1 0

-1 -1 0

0 0 0

0 0 0

2 2 2

Figure2(b)

4 3 3

0 0 -2 0

0 0 0 0

0 0 0 0

0 0 0 0

0 0 3 0

0 0 0 0

0 0 0 0

-1 0 0 0

0 0 -1 0

0 0 0

2 2 2

Sample Output

Figure2(a)

0.0000

(0,0,0) (1,0,0) (2,0,0) (2,1,0)

(2,2,0) (2,2,1) (2,2,2)

Figure2(b)
-1

Problem 3

Consumer Behavior Study

Input file: data3.txt

A major shopping complex opens in Singapore recently. As the operating company is new to the region, it wants to study the buying behavior of consumers. You are appointed as a data analyst by the company to collect statistics on the purchasing pattern of consumers, and to present statistical evidence from the data to be collected from sales counters over a period. In particular, the company is interested in finding the set of items (goods) purchased in one transaction by customers.

The company keeps a log of all the transactions made by consumers over a period of time. Each transaction consists of a set of items (goods) purchased by a consumer. (For simplicity, we ignore item quantity, price and other information from the transaction.) Given the transaction log, the company wants to determine the collection of *itemsets* that meet a certain *support factor threshold*. Each itemset informs the company of the set of items that a consumer frequently buys together.

An itemset X is a subset of a transaction in the transaction log. The support factor of X is the ratio A/B (between 0 and 1) where A is the number of times that X occurs as a subset among all the transactions in the log, and B is the total number of transactions in the log. The company is interested in those itemsets whose support factor is greater than or equal to a certain threshold.

For practical purpose, we assume that the total number of items sold by the company is no more than 100, and that the maximum number of items that can be purchased in a single transaction is limited to 10.

Input

The input file consists of lines of transactions followed by lines of support factor thresholds. Each line of transaction represents the set of items purchased by a customer in a single transaction. The format is as follows:

<Item-1, Item-2, . . . , Item-n>

Each item (which is of no more than 10 characters) in a transaction is separated by a comma, except the last one. The items in a transaction are ordered lexicographically. It is further assumed that no item will repeat itself in a single transaction.

The different support factor thresholds follow the transactions after an empty line. Each threshold is a value between 0 and 1.

Output

For each support factor threshold given in the input file, your program should produce the itemsets whose support factor is greater than or equal to the threshold. The format is as follows: For each threshold, the first outline line displays the threshold value. Subsequent lines display

itemsets (exceeding the threshold) in increasing order of their size in the following format: Each line begins with a (increasing) count value followed by a dot (.) symbol, an itemset, a dash (-) symbol, then the support factor (in brackets) of the itemset at two precision values. Note that items in the itemset are comma-separated and must be lexicographically ordered when displayed. When itemsets are of the same size, they must be lexicographically ordered when displayed.

If there is no itemset for a support factor threshold, your program should display only the heading followed by a blank line.

Sample Input

```
soap,toothbrush,toothpaste
aabattery,wristwatch
shoe,soap,tie,toothpaste
aabattery,toothbrush,toothpaste
```

```
0.25
0.50
```

Sample Output

Relationships for support factor 0.25:

```
1.aabattery,toothbrush-(0.25)
2.aabattery,toothpaste-(0.25)
3.aabattery,wristwatch-(0.25)
4.shoe,soap-(0.25)
5.shoe,tie-(0.25)
6.shoe,toothpaste-(0.25)
7.soap,tie-(0.25)
8.soap,toothbrush-(0.25)
9.soap,toothpaste-(0.50)
10.tie,toothpaste-(0.25)
11.toothbrush,toothpaste-(0.50)
12.aabattery,toothbrush,toothpaste-(0.25)
13.shoe,soap,tie-(0.25)
14.shoe,soap,toothpaste-(0.25)
15.shoe,tie,toothpaste-(0.25)
16.soap,tie,toothpaste-(0.25)
17.soap,toothbrush,toothpaste-(0.25)
18.shoe,soap,tie,toothpaste-(0.25)
```

Relationship for support factor 0.50:

```
1.soap,toothpaste-(0.50)
2.toothbrush,toothpaste-(0.50)
```

Problem 4

Packaging

Input file: data4.txt

A factory produces products that are packed in square packets. All packets have the same height h but are of different sizes 1×1 , 2×2 , 3×3 , 4×4 , 5×5 or 6×6 . These products are always delivered to customers in square boxes of height h and of size 6×6 . In order to reduce transportation cost, it is in the interest of the factory as well as of the customers to minimize the numbers of boxes necessary to deliver the ordered products from the factory to the customer. A good algorithm to resolve the problem of determining the minimum number of boxes necessary to deliver the given products according to a purchase order would save a lot of money. You are asked to write such a program.

Input

The input file consists of lines of purchase order specifications. Each line specifies one purchase order. Orders are described by six integers representing the quantity purchased of packets of size in the order 1×1 to 6×6 . A line containing six zeros indicates the end of the input file.

Output

The output contains one line for each line in the input file. This line specifies the minimum number of boxes that the purchase order from the corresponding line of the input file needs. There is no line in the output corresponding to the last *null* line of the input file.

Sample Input

```
0 0 4 0 0 1
7 5 1 0 0 0
0 0 0 0 0 0
```

Sample Output

```
2
1
```

Problem 5

Drug Tests in Seoul Olympics: To Test or not To Test?

Input file: data5.txt

Should athletes be required to undergo drug testing? And if drug testing is required, should an athlete who tests positive be banned from sport? These questions are haunting the Seoul Olympics Drug Free Association (SODFA) for the past few months. Several sources of uncertainty, as well as several “costs” are involved here.

The uncertainties involve the proportion of the total athlete population who use drugs and the reliability of the tests. If an athlete is tested for a certain type of drug usage (steroids, say), then the test will come out either positive or negative. However, these tests are never perfect. Some athletes who are drug-free tested positive, and some who are drug users tested negative. The former is called *false positives*, and the latter are called *false negatives*. For example, in the last Sydney Olympics, SODFA found out that 5% of all athletes use drugs, 3% of all tests on drug-free athletes yield false positives, and 7% of all tests on drug users yield false negatives.

There are certain “monetary” values involved with drug testing. These values can be broadly categorized into benefits and costs. The benefit for drug testing is correctly identifying a drug user and banning him or her from sports. The costs includes the obvious cost of the tests, but also the less obvious “costs” of invading an athlete’s privacy and of declaring an athlete a drug user when in fact he or she is not (because of faulty test). Specifically, for this problem the monetary values involved are as follows:

- The benefit B from correctly identifying a drug user and barring him or her from sports.
- The cost C1 of the test itself for a single athlete (materials and labor).
- The cost C2 of falsely accusing a non-user and barring him or her from sports.
- The cost C3 of not identifying a drug user (either by not testing at all or by obtaining a false negative).
- The cost C4 of violating a nonuser's privacy by performing the test.

Observe that only C1 is a direct monetary cost that is easy to measure. However, the other “costs” and benefit B are real, and they must be compared on the same scale to enable administrators to make a rational decision. In this problem you must do so by comparing everything to the cost C1. (We would express all other costs as multiples of C1.)

You may assume that there are only two alternatives: perform drug testing on all athletes or do not perform any drug testing. In the former case, the result of the drug test is first observed. Each test result leads to an action (ban or no ban from sports), and then the eventual benefit or cost depends on whether the athlete uses drugs. If no drug testing is performed, then there is no intermediate test result. You may also assume that all benefits have a positive sign and all costs have a negative sign associated with them.

SODFA needs your help to write a program that computes:

- The probability that an athlete who tested positive is a drug user.
- The probability that an athlete who tested negative is drug-free.

- The *Potential Monetary Value* for performing drug testing (denoted by PMV_T). It is defined as: $PMV_T = W_+p_+ + W_-p_-$ where W_+ and W_- are the *weighted costs* of the results of a drug test (positive and negative) and p_+ and p_- are their probabilities respectively. The weighted cost of a positive or negative drug test result is a weighted average of the total monetary values associated with the *eventual* outcomes of the drug test result, weighted by their probabilities. Note that an *eventual* outcome of a positive drug test may yield a false positive. Similarly, an eventual outcome of a negative drug test may yield a false negative. Formally, if v_i is the total monetary value corresponding to eventual outcome i and p_i is its probability, then W is defined as follows:

$$W = \sum v_i p_i$$

- The *Potential Monetary Value* for not performing drug testing (denoted by PMV_{NT}). It is a weighted average of the total monetary values associated with the eventual outcomes for not performing drug test, weighted by their probabilities. The eventual outcome in this case may yield a non-user or a drug user who escapes from being barred from sports. Formally, if v_j is the monetary value corresponding to eventual outcome j and p_j is its probability then PMV_{NT} is defined as follows:

$$PMV_{NT} = \sum v_j p_j$$

Input

The input consists of nine lines. The first line contains an integer and represents the percentage of an athlete's using drugs. The second and third lines also contain integers representing the percentages of all tests that yield false positives and false negatives respectively. Each of the remaining lines contains a string and an integer that denotes different types of monetary values and their values. The integers are always positive. The last line terminates the input and should not be processed.

Output

For the given values in the input file, output the following:

- The probability that an athlete who tested positive is a drug user. Print the string "Positive =" and the value of the probability accurate to four decimal places.
- The probability that an athlete who tested negative is drug-free. Print the string "Negative =" and the value of the probability accurate to four decimal places.
- The PMV for performing a drug test. Print the string "PMV-Test =" and the value accurate to two decimal places.
- The PMV for not performing a drug test. Print the string "PMV-Not-Test =" and the value accurate to two decimal places.
- Print a blank line between each output.

Sample Input

5
3

7
25 B
1 C1
50 C2
20 C3
2 C4
0 END

Sample Output

Positive = 0.6200
Negative = 0.9962
PMV-Test = -3.23
PMV-Not-Test = -1.00

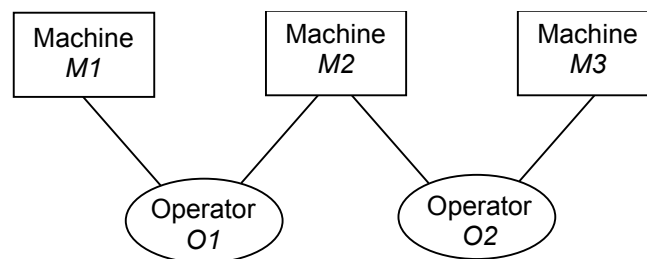
Problem 6

Conflict Sets in Resource Sharing

Input file: data6.txt

In a manufacturing simulation, a *process route* specifies the sequence of various processing steps required to transform raw materials into finished products. Processing steps are arranged in the order in which they occur in the manufacturing of the product. Each processing step is represented by a record that contains information for that step. The record includes the *machine* and *operators* required for the step and the *processing time*. For example, test case 1 in the sample input specifies a process route with five processing steps. Machine *M1* and operator *O1* are used in the first step which requires a processing time of 0.1 time unit. Exactly one machine is used per processing step. However, zero, one or multiple operators may be required in a given processing step.

It is possible for two different machines to require the same operator to perform the operation (e.g., both processing steps 1 and 3 in test case 1 require operator *O1* and both processing steps 3 and 4 require operator *O2*). In addition, different operators may be required to operate the machine in a given processing step (e.g., processing steps 3 in test case 1 uses both *O1* and *O2*). This leads to situations as shown in the following figure, where machine *M1* and *M2* share operator *O1*, and machine *M2* and *M3* share operator *O2*.



Machines and operators are modeled as resources in simulation. The relationship between machines and operators can be defined as follows: machine *X* and operator *Y* are related to each other (written as $X \leftrightarrow Y$) if *X* and *Y* are required in the same processing step. Note that relation \leftrightarrow is symmetric. An equivalence class can then be defined by the transitive closure of this relation (written as \leftrightarrow^+). For resources *X* and *Y*,

$X \leftrightarrow^+ Y$, if $X \leftrightarrow Y$; or

$X \leftrightarrow^+ Y$, if there exists a resource *Z* and $X \leftrightarrow^+ Z$, then $Z \leftrightarrow^+ Y$.

The edges shown in the above graph represent relation \leftrightarrow . Any two resources connected by a path must be in the same equivalence class.

To easily resolve the resource requirements in the simulation, it requires that machines in the same equivalence class are put together to form a *conflict set*. Thus, in the above example, *M1*, *M2* and *M3* form a conflict set.

In this problem, you are requested to develop a program that generates the following information from a process route:

1. *Conflict Sets*: Each conflict set consists of a machine or a group of machines according to the above definition. For test case 1 in the sample input, there are two conflict sets: $CS1$ and $CS2$. $CS1 = \{M1, M2, M3\}$ and $CS2 = \{M4\}$.
2. *Connectivity between Conflict Sets*: There is a direct link from conflict set CS' to conflict set CS'' if there exist machines $M' \in CS'$ and $M'' \in CS''$ so that M' and M'' are used in two consecutive processing steps. For test case 1, there is a link from $CS1$ to $CS2$, since $M1 \in CS1$, $M4 \in CS2$, and $M1$ and $M4$ are used in two consecutive processing steps (i.e., steps 1 and 2). Similarly, there is also a link from $CS2$ to $CS1$.
3. *Weight of Conflict Set*: The weight of a conflict set is defined as the aggregated processing time of all the steps involved by the machines in the conflict set. For example, the weight of conflict set $CS1$ is 2.6 which is the summation of processing times at steps 1, 3, 4 and 5 (i.e., all steps involved by machines $M1$, $M2$ and $M3$).

Input

The input consists of a number of test cases, each of which describes a process route and ends with a # sign. A process route consists of a number of lines, each of which describes the machine, operators and processing time required for a processing step. It is given in the following format:

step_id machine_id list_of_operator_ids processing_time

Step identifiers are specified using integers. Identifiers of machines and operators are specified using integers prefixed by characters "M" and "O" respectively, and processing times are specified using floating-point numbers (with a single decimal place). Note that there is exactly one machine per processing step. However, zero, one or multiple operators may be required at a given processing step.

Output

The definition of the conflict sets should be printed according to the following format (see the sample output below):

Total number of conflict sets

List of machine identifiers in each conflict sets

The connectivity between conflict sets and the weight of conflict set should be printed as a *connectivity matrix* according to the following definition (see the sample output below):

CM is a connectivity matrix. $CM(i, i)$ contains the weight of conflict set i . $CM(i, j)$, $i \neq j$, contains 1 if there is a direct link from conflict set i to conflict set j ; otherwise it contains 0.

Sample Input

```
1 M1 O1 0.2
2 M4 O3 1.0
3 M2 O1 O2 1.2
4 M3 O2 0.2
```

5 M1 1.0

1 M1 O1 0.5
2 M2 O1 0.5
3 M3 1.0
#

Sample Output

Test Case 1:
Total number of conflict sets: 2
Conflict Set 1: M1, M2, M3
Conflict Set 2: M4
Connectivity Matrix:
2.6, 1
1, 1.0

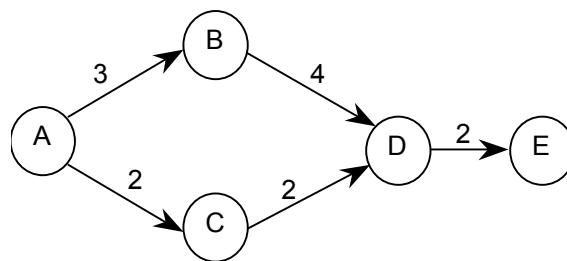
Test Case 2
Total number of conflict sets: 2
Conflict Set 1: M1, M2
Conflict Set 2: M3
Connectivity Matrix
1.0 1
0 1.0

Problem 7

Activity Scheduling in Project Management

Input file: data7.txt

Mr. John Chong is a project manager in NewAge.com Pte Ltd. His task is to schedule activities in a project to make sure that the project can be completed in a minimum amount of time. An *activity* is a part of the project that takes place over a period of time and a *milestone* is the completion of an activity (or a group of activities). Activities in a project are usually specified using an *activity graph* as shown below. Nodes in the activity graph are milestones and links are activities. The label of a link indicates the duration of the corresponding activity (in the unit of days). An activity can be specified using a *starting* and an *ending milestone*. For example, activity *BD* takes 4 days to complete. It starts from milestone *B* and ends at milestone *D*. Each activity graph has exactly one *start* and one *finish milestone*. In the following diagram, the start milestone is *A* and the finish milestone is *E*.



An activity graph also depicts the dependencies among activities and milestones. An activity starting from a milestone can only begin after all activities ending at that milestone have been completed. For instance, activity *DE* can only begin after activities *BD* and *CD* have been completed.

Given an activity graph, the minimum amount of time it will take to complete the project can be easily calculated. This is referred to as the *minimum duration* of the project. The minimum duration of the project depicted by the above activity graph is 9 days. It is determined by the activities *AB*, *BD* and *DE*. However, besides minimum duration, John also needs to know:

1. the earliest time an activity may begin (referred to as *earliest starting time*, EST, of an activity), and
2. the latest time an activity may begin so that the project may still be completed in minimum duration (referred to as *latest starting time*, LST, of an activity).

For example, in the above activity graph, activities *AB* and *AC* can begin immediately since they do not depend on any other activities. So, their EST is 1 (i.e., day 1). Activity *BD* can only begin after activity *AB* has been completed. Its EST, therefore, is 4. Similarly, the EST of activity *CD* is 3. Activity *DE* can only begin after activities *BD* and *CD* have been completed. Since *BD* cannot be completed until day 8 and *CD* cannot be completed until day 6, the earliest starting time for activity *DE* is 8.

In order to complete the project in minimum duration, activity *DE* then must begin at day 8. So, its LST is 8. For activity *DE* begins at day 8, activity *BD* has to start at day 4. However, activity *CD* can begin as late as day 6 without delaying the schedule of activity *DE*. So, LSTs of activities *BD* and *CD* are 4 and 6 respectively. Similarly, LSTs of activities *AB* and *AC* can be calculated. They are 1 and 4 respectively.

In this problem, you are requested to develop a program to help John schedule activities in a project. Activities and their duration are specified in the input. Your program should output the EST and LST of each activity and the list of activities that determine the minimum duration of the project.

Input

The input consists of a number of test cases, each of which ends with a # sign. A test case describes an activity graph. It consists of a start and a finish milestone of the activity graph and a list of activities. Each activity is specified in the following format by a starting milestone, an ending milestone and its duration:

starting_milestone ending_milestone duration

Milestones are specified using alphabet characters and duration is specified using integers.

Output

For each test case, the EST and LST of each activity should be printed out in the following format (see the sample output below):

starting_milestone ending_milestone EST LST

Activities that determine the minimum duration of the project should also be printed out (see the sample output below).

Sample Input

```
A
E
A B 3
A C 2
B D 4
C D 2
D E 2
#
A
F
A B 2
A C 3
B D 3
B E 2
C D 1
C E 3
D F 4
```

E F 2

#

Sample Output

Test Case 1

A B 1 1

A C 1 4

B D 4 4

C D 3 6

D E 8 8

Activities that determine minimum duration

A B

B D

D E

Test Case 2

A B 1 1

A C 1 2

B D 3 3

B E 3 6

C D 4 5

C E 4 5

D F 6 6

E F 7 8

Activities that determine minimum duration

A B

B D

D F

Problem 8

Expert systems

Input file: data8.txt

An expert system consists of two main components. The *knowledge base* contains the knowledge with which the *inference engine* draws conclusion. In this problem, you will build a very small expert system. Each *knowledge* (or *fact*) is represented by a certain predicate name along with two arguments. Some possible facts are shown below:

father_of (tom, john) . (1)
mother_of (mary, john) . (2)
child_of (susan, mary) . (3)

The first fact states that Tom is the father of John. The second fact states that Mary is the mother of John, and the third fact states that Susan is a child of Mary. Each rule in the inference engine is represented in the following format:

$p :- q .$

where p is the rule's *head* and q is the *subgoal*. This expression states that the head goal p is satisfied if the subgoal q is satisfied. As an example, the "parent_of" rules are as follows:

parent_of (X,Y) :- father_of (X,Y) . (4)
parent_of (X,Y) :- mother_of (X,Y) . (5)
parent_of (X,Y) :- child_of (Y,X) . (6)

The rules state that X is the parent of Y if X is the father of Y, X is the mother of Y, or Y is a child of X. Note that the name of the head might be the same as the name of the subgoal.

There are two types of queries that a user may raise in this system. The first type of queries is similar to the statement of a fact, except that one argument is a *variable* (that is, the unknown argument), and it begins with an uppercase letter. The system will need to use the facts it has to compute all the possible values of the variable. For example, the following query asks who is John's parent:

parent_of (X, john) .

The system yields the following answers based on facts (2) and (1):

X = mary
X = tom

Note that there might be one or several values associated with X. If no proper value is found, there will be no output. The second type of queries looks like a rule with an empty head, which might yield an output of TRUE or FALSE. An example is as follows:

```
:- parent_of (tom, john) .
```

This query asks if Tom is the parent of John. In this case, the system yields the following answer based on fact (1) and rule (4):

```
TRUE
```

Input

The input file consists of facts, rules, and queries. Each statement terminates with a period and there are less than 100 statements in the input file. As a convention, constants begin with a lowercase letter and variables begin with an uppercase letter. A single “0” symbol in the input line signifies the end of input.

Output

Answer each query by providing the value of the variables, or TRUE or FALSE. Note that the answer is based on the facts/rules presented before the queries, and not on all the facts/rules in the file. If there are several answers for a query, remove any duplicates and sort the results in alphabetical order. Each line should contain only one single answer. Draw a dashed line like “----” after the output of each query. Use a single “*” symbol to signify the end of output.

Sample Input

```
father_of (tom, john) .  
parent_of (X, Y) :- father_of (X, Y) .  
parent_of (X, Y) :- mother_of (X, Y) .  
parent_of (X, Y) :- child_of (Y, X) .  
parent_of (X, john) .  
mother_of (X, john) .  
:- parent_of (mary, john) .  
mother_of (mary, john) .  
parent_of (X, john) .  
child_of (susan, mary) .  
parent_of (mary, X) .  
0
```

Sample Output

```
X = tom  
----  
----  
FALSE  
----  
X = mary  
X = tom  
----  
X = john  
X = susan
```

*