

ICPC Practice Set 3

Problem 1 Empire of Agstyan

The emperor of Agstyan has recently died, and many of his generals have partitioned the country up between themselves. Now, each of them wants to become the new emperor, and will fight with the other generals until only a single winner remains. As advisor to a nearby kingdom, you are asked to advise the king on which general is going to become the new emperor, so that your king can try to take his side in the war.

Agstyan is a rectangular country, and all of the land is divided up into squares. Each general controls a contiguous region of squares. Between any two or more generals' areas, there is a "no-man's land" (NML) where neither general controls the territory. The strength of a general's army is directly related to the numbers of squares the general controls. That is, if a general controls 25 squares, then his army has strength 25.

Each month, each general attacks at most one neighboring general with a weaker army. The general with the stronger army wins and takes over the territory belonging to that general. In addition (unless the general has also lost his own area), any NML squares no longer needed become part of that general's territory. This process continues until one general controls all of Agstyan.

Some further details on the problem:

- Neighboring squares are those to the North, South, East, and West (neighboring connections are not made to the NE, SE, NW, SW)
- Two generals are neighboring if they have neighbors in either the N, S, E, or W direction with exactly one "no-man's land" square in between
- Generals are named by letters of the alphabet, from "A" to "Z" (there are at most 26 generals)
- If two generals have the same strength armies, then the one that comes first alphabetically is considered stronger
- Generals always attack the weakest neighboring general
- If no neighbors are weaker, the general does not attack, but just waits (and might get taken over)
- It is possible for a general to attack another general and take over his land, but also to be attacked and lose his original land to another general. The general still controls the newly captured territory, and is not (yet) eliminated.
- If two or more generals attack the same other general in a month, the strongest general among all of them wins.
- All generals' territory is contiguous.
- At the beginning, there may be extra NML squares on the board. These should be eliminated as they become surrounded by a single general.
- When one general takes over another's territory, and the general maintains his own land, any NML that formed a border between the two generals, and is not needed to form a border with any other general, can be eliminated (becomes the property of the winning general).
- At the end of any month, there should be no NML squares that can be eliminated. Squares can be eliminated if a group of squares is surrounded by the same general (and any map edges)
- Though it is possible to create maps where generals cannot attack each other (due to extra NML spaces), there will always be a single winning general

Input:

The input consists of a series of possible maps of Agstyan. The first line of each map gives the number of squares in width (w) and height (h) of the map, where w and h are between 1 and 25, inclusive. The next h lines each contain w characters, describing the starting positions of the generals. Each square is labeled either with a letter from 'A' to 'Z' (the general's name controlling that square) or a '*' indicating the square is part of a no-man's land. Input is terminated by a map of size 0 by 0.

Output:

For each valid input map, you are to output a single line saying: "# is the emperor." where # is the name of the general.

Example Input:

```
7 8
A**DD*C
A**DD**
A**D*EE
A****EE
AAA****
**A*F*B
G*AA*BB
****BBB
0 0
```

Example Output:

A is the emperor.

Illustration:

To illustrate the above example, the state of the empire at the end of each month is shown below:

```
A**DD*C      A**DDDD      A**DDDD      A**DDDD      AAAAAAA
A**DD**      A**DD**      A**DD**      A**DD**      AAAAAAA
A**D*EE      A**D*EE      A**D*BB      A**D*AA      AAAAAAA
A****EE      A****EE      A****BB      A****AA      AAAAAAA
AAA****      AAA****      AAA****      AAAAAAA      AAAAAAA
**A*F*B      AAA*BBB      AAAAAAA      AAAAAAA      AAAAAAA
G*AA*BB      AAAA*BB      AAAAAAA      AAAAAAA      AAAAAAA
****BBB      ****BBB      AAAAAAA      AAAAAAA      AAAAAAA
```

So, in the first month, A attacks G (it could have attacked F or B, also, but G was weakest), B attacks F (which was weaker than E), and both D and E attack C (with D winning). When A attacks G, two NML squares that formed borders are converted to A's territory, and one that is then surrounded by A becomes A's territory. Both B and D pick up one NML square that formed a border.

In the second month, A attacks B (taking its territory and two NML border squares plus four NML surrounded squares) while D attacks E (taking its territory). Both B and D attack E (with B winning), so B takes over E's territory.

In the third month, both A and D attack B, with A winning.

Finally, in the fourth month, A takes over D, conquering all of Agstyan

Problem 2 DNA Computer

DNA is made up of four different nucleotides, adenine, thiamine, guanine, and cytosine, usually abbreviated with the letters ATGC. DNA is formed in a helix, where two complementary chains of nucleotides meet with each other. Each A is paired with a T, and each G with a C. Thus, the following two sequences would be pairs, for example:

```
AAAGGTCCGATC
TTTCCAGGCTAG
```

A DNA computer program is written as a single sequence of nucleotides, which encodes the necessary information. It is important that this sequence be allowed to bind with a complementary sequence. Unfortunately, some sequences can “bind” with themselves, creating problems in actually using that code sequence. For example, the following sequence will bind with itself, as shown:

AAAGGTCCGATCATGCCAGTTTCAGGACCTACCTTC, binding as:

```
AAAGGTCCGATCATGCC
  TTCCA                A
C      T                G
          CCAGGACTTT
```

You are to write a program that will determine whether a sequence of code will be self-binding. For the purpose of this program, to be self-binding, the code must have all of the following properties:

- A sequence of at least 5 consecutive nucleotide pairs that match up (either in the same order or in reverse order) with a sequence in another part of the code.
- If the matching sequence is in the “reverse” order (as in the sequence above), there must be at least 5 nucleotide pairs between the two sections for the sequence to be able to “loop back” on itself. For example, the following would be self binding:
AAATTXXXXAATTT (where the X could be anything), but the following would not:
AAATTXXXXAATTT
- If the matching sequence is in the “forward” order, there must be at least 10 nucleotide pairs between the two sections for the sequence to be able to bind to itself. For example, the following could self-bind: AAATTXXXXXXXXXXTTTAA but the following could not: AAATTXXXXTTTAA
- If a sequence will match in either reverse or forward order, only 5 pairs need to intervene. For example, AAAAAXXXXXTTTTT would self-bind.

Input:

The input will contain a series of DNA sequences. Each sequence will consist of a series of letters, A, T, C, or G. Each line will contain one sequence, with a maximum of 75 nucleotides in each sequence. The final line (which should not be processed) will consist of a single letter, E.

Output:

You are to output one line for each sequence. The line should say one of the two following things (the X is the sequence number):

Sequence X can self bind.

Sequence X can not self bind.

Example Input:

```
AAAAAAAAAAAA
AAAAACGCGCTTTT
AAATTCGCGCTTTAA
AAATTCGCGCAATTT
E
```

Example Output:

Sequence 1 can not self bind.

Sequence 2 can self bind.

Sequence 3 can not self bind.

Sequence 4 can self bind.

Problem 3 Interval Calculations

Interval arithmetic is used to represent numbers that are known imprecisely, and to keep track of error in computations. A number, x , is represented as an interval: $[a,b]$, which means that $a \leq x \leq b$. Notice that a number known exactly can still be represented as an interval, e.g. $3 = [3,3]$. Basic math can be performed on these numbers. For the purpose of this question, we will deal only with addition and subtraction. These are defined by the following equations:

$$[a,b] + [c,d] = [a+b,c+d]$$

$$[a,b] - [c,d] = [a-d, b-c]$$

Comparisons between numbers can also be made. For example, $[a,b] < [c,d]$ if and only if $b < c$.

Two numbers can only be equal if they are exactly the same, e.g. $[3,3] = [3,3]$.

An interval representation means that a number can be any value in that interval, so if $a = [1,2]$ and $b = [1,2]$, then we cannot compare a and b , since, for example the “true” value of a could be 1.1, and the “true” value of b could be 1.6, or vice-versa.

You are to write a program that processes a short sequence of operations on numbers given as intervals, and determines what can be determined about the numbers.

Input:

The input will consist of a number of lines, each containing either an assignment or a query. You are to read input until a line consisting of just the number 0 is reached. Assignment lines and query lines can come in any order or mixture. Assignment lines can take one of three forms:

- $A = B$
- $A = B + C$
- $A = B - C$

In each case, B or C may be one of three possible forms:

- A variable written as a single lower-case character from ‘a’ to ‘z’
- An exact number (e.g. 3)
- An interval (e.g. $[1,2]$)

A is always a variable. For example, the following are possible assignments:

$$d = [1, 2] + [2, 2]$$

$$q = [1, 2] - 2$$

$$e = d$$

$$z = z + x$$

Query lines always begin with a variable, followed by a ‘?’, followed by the thing to compare to. This last entry can be either a variable, an exact number, or an interval. For example, the following are possible queries:

$$d ? 5$$

$$e ? a$$

$$f ? [2, 3]$$

You can assume that any variables used in a query or on the right hand side of an assignment will have been previously assigned. That is, you could not have a query such as $a ? b$ unless both a and b had been defined above. You may assume that all intervals are valid (i.e. for an interval $[a,b]$, $a \leq b$). Treat any variable on the right hand side by its value, rather than as a variable.

Output:

Your output should consist of one line for each query. The line should be the same as the line in the input file, but the '?' replaced by the appropriate symbol: '>', '>=', '=', '<=', '<', or '?'. You are to use a '?' if and only if it is not possible to determine the relationship. Variables should have no "memory" – that is, if a variable is an interval, it should have no knowledge of how that interval was formed.

The following gives an example input and output file. Notice that the final query demonstrates the lack of memory property (i.e. even though we would know that $a=b$, we have no memory of that when performing the query):

Example Input:

```
c = 3
b = [1, 2]
a = b + c
a = a - 2
a ? c
b ? a
b = [3, 3]
b ? c
c = [1, 4]
a ? c
a ? [-2, -1]
a = [1, 2]
b = a
a ? b
0
```

Example Output:

```
a <= c
b <= a
b = c
a ? c
a > [-2, -1]
a ? b
```

Problem 4 Encyclopedia

You are in charge of laying out the order of a print encyclopedia. Each entry (article) in the encyclopedia is written by a different author. Your job is to group the articles alphabetically and arrange them into volumes. Each volume has a maximum number of pages.

Unfortunately, not all authors have submitted articles at the same time. The cutoff date was 9/30, and some submitted the articles before then, some after then. You have decided that you will guarantee that all articles submitted on 9/30 or before will make the encyclopedia. Volumes will be decided based only on these articles.

Articles submitted on 10/1 or later will only be included if there are enough extra pages available in the appropriate volume to take that article. These later articles will be added (or rejected) in a first-come, first-served manner, with articles arriving on the same day being added (or rejected) in alphabetical order. If a new article falls between two volumes of the dictionary, you should first try to add it to the earlier volume, and if that cannot be done (i.e. there are not enough pages left for it in that volume), try to add it to the subsequent one.

Input:

The first line of input will give the maximum number of pages in one volume of the encyclopedia. You can assume that no article will be larger than this size. Each following line of input data will contain information on one article, in this order:

<subject> - a string of not more than 20 characters. The first character will always be capitalized, and following characters will be lower-case. There will be no spaces

<# of pages> - a positive integer

<date submitted> - a date in the form MM/DD where MM is the month and DD the day.

These lines will not be given in any particular order. You can stop reading data when the word End is read.

Output:

You are to output the following information. The first line of the output should state:

X volumes were produced.

(where X is the number of volumes used). The X subsequent lines should each say:

Volume Y has Z pages.

(where Y is the volume number and Z is the number of pages in that volume).

Example Input:

```
20
Algol 5 09/29
Turing 9 10/01
C 4 10/02
Ada 10 09/29
Fortran 10 10/01
Scheme 6 09/28
Cobol 15 09/30
End
```

Example Output:

```
3 volumes were produced.
Volume 1 has 19 pages.
Volume 2 has 15 pages.
Volume 3 has 16 pages.
```

Although you did not have to print this out, the first volume would contain entries for Ada, Algol, and C, the second volume would contain an entry for Cobol, and the third volume would contain entries for Fortran and Scheme. Turing would not have been included.

Problem 5 Card Counter

Blackjack players can use card counting techniques to determine when a particular deck of cards is “hot” (in a state such that the advantage is on the player’s side, rather than on the casino’s). You are to write a program to help determine whether a deck is hot.

A deck of cards has 4 “suits” with 13 cards in each suit. The cards are numbered from 2 – 10, in addition to the Ace (can be considered a 1 or an 11), the King, the Queen, and the Jack. We will consider the King, Queen, Jack, and Ten to be “high” cards.

You are to take in a list of cards that have already been drawn/used, and try to determine if the deck is “hot.” We will say it is hot if the chance of drawing a high card on the next card is at least 5% greater than the percentage in an initial deck.

Note that to help combat card counting techniques, casinos will usually use several decks at once.

Input:

The input consists of data from a series of blackjack tables. The first line of the input gives the number of standard (52-card) decks being combined into one at that table. A table with 0 decks indicates the end of the input. Then, subsequent lines will list the specific cards already drawn at that table. Cards will be separated either by a space (if on one line) or by line breaks. Suits are ignored, and the letters A, J, Q, and K will be used to refer to the Ace, Jack, Queen, and King. The end of the list is determined by a card labeled E.

Output:

If the deck is not hot, output “Deck # is not hot.” If it is hot, output “Deck # is hot!” In both cases, the # refers to the table number, starting at 1, in the order they appeared.

Example Input:

```
1
2 9 A 2 5 8 4 8 6 7 E
2
2 9 A 2 5 8 4 8 6 7 E
1
2 9 A 2 5
8 4 8
6 7
E
0
```

Example Output:

```
Deck 1 is hot!
Deck 2 is not hot.
Deck 3 is hot!
```

Problem 6 Adjective Grouping

You are trying to list words that certain adjectives apply to. You have collected various adjective-noun pairs (like “brown cow”, “fast car”, etc.). You want to produce a list of adjectives, and for each adjective, a list of nouns that fits it.

Input:

The input file consists of a list of adjective-noun pairs. The adjective is always first, and the noun is always second. Both the noun and adjective will be single words composed entirely of lower-case letters. No pairs will be repeated. Continue reading pairs until the words “stop now” are read.

Output:

For each adjective encountered, output two lines. The first line should be the adjective followed by a colon (':'). The second line should list all nouns that adjective has been applied to in the list, separated by a space. Adjectives should be output in the order they were first encountered in the input, and nouns should be output in the order they were encountered in the input.

Example Input:

```
hot tamale
hot pan
red truck
fast car
fast cheetah
hot water
blue sky
hot sidewalk
blue moon
```

Example Output:

```
hot:
tamale pan water sidewalk
red:
truck
fast:
car cheetah
blue:
sky moon
```