Problem A: String Computer

Extel have just brought out their newest computer, a string processing computer dubbed the X9091. It is hoped that it will have some value in cryptography and related fields. (It is rumoured that the Taiwanese are working on a clone that will correct Stage 1 essays, but we will ignore such vapour-ware). This computer will accept input strings and produce output strings from them, depending on the programs loaded into them at the time. The chip is the ultimate in RISC technology—it has only three transformation instructions:

- Delete a character at a particular position.
- Insert a character at a particular position.
- Change the character at a particular position to a different character.

Programs for this machine are written in a form of machine code where each instruction has the format ZXdd—Z represents the code for the instruction (D, I or C), X is a character and dd represents a two digit number. A program is terminated by a special halt instruction consisting of the letter 'E'. Note that each instruction works on the string in memory at the time the instruction is executed.

To see how this all works consider the following example. It is desired to transform the string 'abcde' to the string 'bcgfe'. This could be achieved by a series of Change commands, but is not minimal. The following program is better.

```
      abcde
Da01  bcde   % note the 'a' is necessary because it is checked by the hardware
Cg03  bcge
If04  bcgfe
E     bcgfe  % Terminates the program
```

Write a program that will read in two strings (the input string and the target string) and will produce a minimal X9091 program necessary to transform the input string into the target string. Since there may be multiple solutions, only one should be produced. Any solution that satisfies these criteria will be accepted.

Input will consist of a series of lines, each line containing two strings separated by exactly one space. The strings will consist of no more than 20 lower case characters. The file will be terminated by a line consisting of a single #.

Output will consist of a series of lines, one for each line of the input. Each will consist of a program in X9091 language.

Sample input

```
abcde bcgfe
#
```

Sample output

```
Da01Cg03If04E
```

Division 1

Problem B Stamps

The government of Nova Mareterrania requires that various legal documents have stamps attached to them so that the government can derive revenue from them. In terms of recent legislation, each class of document is limited in the number of stamps that may be attached to it. The government wishes to know how many different stamps, and of what values, they need to print to allow the widest choice of values to be made up under these conditions. Stamps are always valued in units of $1.

This has been analysed by government mathematicians who have derived a formula for n(h,k), where h is the number of stamps that may be attached to a document, k is the number of denominations of stamps available, and n is the largest attainable value in a continuous sequence starting from $1. For instance, if h=3, k=2 and the denominations are $1 and $4, we can make all the values from $1 to $6 (as well as $8, $9 and $12). However with the same values of h and k, but using $1 and $3 stamps we can make all the values from $1 to $7 (as well as $9). This is maximal, so n(3,2) = 7.

Unfortunately the formula relating n(h,k) to h, k and the values of the stamps has been lost — it was published in one of the government reports but no-one can remember which one, and of the three researchers who started to search for the formula, two died of boredom and the third took a job as a lighthouse keeper because it provided more social stimulation.

The task has now been passed on to you. You doubt the existence of a formula in the first place so you decide to write a program that, for given values of h and k, will determine an optimum set of stamps and the value of n(h,k).

Input will be from a file called PROBLEMB.DAT and will consist of several lines, each containing a value for h and k. The file will be terminated by two zeroes (0 0). For technical reasons the sum of h and k is limited to 9. (The President lost his little finger in a shooting accident and cannot count past 9).

Output will consist of a line for each value of h and k consisting of the k stamp values in ascending order right justified in fields 3 characters wide, followed by a space and an arrow (->) and the value of n(h,k) right justified in a field 3 characters wide.

Example

INPUT
```
3 2
0 0
```

OUTPUT
```
  1  3 ->  7
```

Division 1

Problem C — Making Change

Given an amount of money and unlimited (almost) numbers of coins (we will ignore notes for this problem) we know that an amount of money may be made up in a variety of ways. A more interesting problem arises when goods are bought and need to be paid for, with the possibility that change may need to be given. Given the finite resources of most wallets nowadays, we are constrained in the number of ways in which we can make up an amount to pay for our purchases — assuming that we can make up the amount in the first place, but that is another story.

The problem we will be concerned with will be to minimise the number of coins that change hands at such a transaction, given that the shopkeeper has an adequate supply of all coins. (The set of New Zealand coins comprises 5c, 10c, 20c, 50c, $1 and $2.) Thus if we need to pay 55c, and we do not hold a 50c coin, we could pay this as 2*20c + 10c + 5c to make a total of 4 coins. If we tender $1 we will receive 45c in change which also involves 4 coins, but if we tender $1.05 ($1 + 5c), we get 50c change and the total number of coins that changes hands is only 3.

Write a program that will read in the resources available to you and the amount of the purchase and will determine the minimum number of coins that change hands.

Input will be from a file called PROBLEMC.DAT and will consist of a series of lines, each line defining a different situation. Each line will consist of 6 integers representing the numbers of coins available to you in the order given above, followed by a real number representing the value of the transaction, which will always be less than $5.00. The file will be terminated by six zeroes (0 0 0 0 0 0). The total value of the coins will always be sufficient to make up the amount and the amount will always be achievable, i.e. it will always be a multiple of 5c.

Output will consist of a series of lines, one for each situation defined in the input. Each line will consist of the minimum number of coins that change hands right justified in a field 3 characters wide.

Example

INPUT
```
2 4 2 2 1 0  0.95
2 4 2 0 1 0  0.55
0 0 0 0 0 0
```

OUTPUT
```
  2
  3
```

Division 1

Problem D — The Sultan's Successors

The Sultan of Nubia has no children, so she has decided that the country will be split into up to k separate parts on her death and each part will be inherited by whoever performs best at some test. It is possible for any individual to inherit more than one or indeed all of the portions. To ensure that only highly intelligent people eventually become her successors, the Sultan has devised an ingenious test. In a large hall filled with the splash of fountains and the delicate scent of incense have been placed k chessboards. Each chessboard has numbers in the range 1 to 99 written on each square and is supplied with 8 jewelled chess queens. The task facing each potential successor is to place the 8 queens on the chess board in such a way that no queen threatens another one, and so that the numbers on the squares thus selected sum to a number at least as high as one already chosen by the Sultan. (For those unfamiliar with the rules of chess, this implies that each row and column of the board contains exactly one queen, and each diagonal contains no more than one.)

Write a program that will read in the number and details of the chessboards and determine the highest scores possible for each board under these conditions. (You know that the Sultan is both a good chess player and a good mathematician and you suspect that her score is the best attainable.)

Input will be from a file called PROBLEMD.DAT and will consist of k (the number of boards), on a line by itself, followed by k sets of 64 numbers, each set consisting of eight lines of eight numbers. Each number will be a positive integer less than 100. There will never be more than 20 boards.

Output will consist of k numbers consisting of your k scores, each score on a line by itself and right justified in a field 5 characters wide.

Example

INPUT

```
1
 1  2  3  4  5  6  7  8
 9 10 11 12 13 14 15 16
17 18 19 20 21 22 23 24
25 26 27 28 29 30 31 32
33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48
48 50 51 52 53 54 55 56
57 58 59 60 61 62 63 64
```

OUTPUT

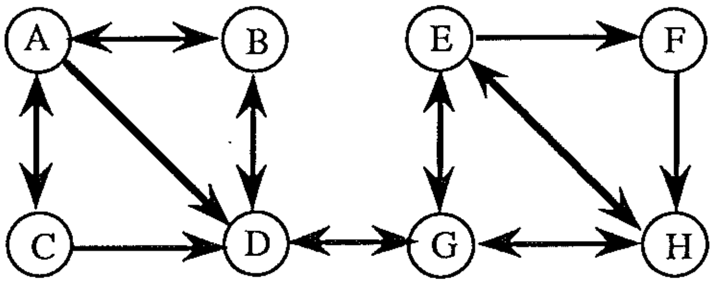
```
  260
```

Problem E Theseus and the Minotaur

Those of you with a classical education may remember the legend of Theseus and the Minotaur. This is an unlikely tale involving a bull headed monster, an underground maze full of twisty little passages all alike, love-lorn damsels and balls of silk. In line with the educational nature of this contest, we will now reveal the true story.

The maze was actually a series of caverns connected by reasonably straight passages, some of which could only be traversed in one direction. In order to trap the Minotaur, Theseus smuggled a large supply of candles into the Labyrinth, as he had discovered that the Minotaur was afraid of light. Theseus wandered around somewhat aimlessly until he heard the Minotaur approaching along a tunnel. At this point he lit a candle and set off in pursuit. The Minotaur retreated into the cavern it had just left and fled by another passage. Theseus followed, slowly gaining, until he reached the k'th cavern since lighting the candle. Here he had enough time to place the lighted candle in the middle of the cavern, light another one from it, and continue the chase. As the chase progressed, a candle was left in each k'th cavern passed through, thereby limiting the movement of the Minotaur. Whenever the Minotaur entered a cavern, it would check the exits in a particular order, fleeing down the first that did not lead directly to a lit cavern. (Remember that as Theseus was carrying a lit candle, the Minotaur never exited a cavern by the tunnel used to enter it.) Eventually the Minotaur became trapped, enabling Theseus to defeat it.

Consider the following Labyrinth as an example, where in this case the Minotaur checks the exits from a cavern in alphabetical order:


$$\begin{tikzcd}
A \arrow[r, leftrightarrow] \arrow[d, leftrightarrow] \arrow[dr] & B \arrow[d, leftrightarrow] & E \arrow[r] \arrow[d, leftrightarrow] \arrow[dr, leftrightarrow] & F \arrow[d] \\
C \arrow[r] & D \arrow[r, leftrightarrow] & G \arrow[r, leftrightarrow] & H
\end{tikzcd}$$

Assume that Theseus is in cavern C when he hears the Minotaur approaching from A, and that for this scenario, the value of k is 3. He lights a candle and gives chase, pursuing it through A, B, D (leaves a candle), G, E, F (another candle), H, E, G (another), H, E (trapped).

Write a program that will simulate Theseus's pursuit of the Minotaur. The description of a labyrinth will identify each cavern by an upper case character and will list the caverns reachable from that cavern in the order that the Minotaur will attempt them, followed by the identifiers for the caverns which the Minotaur and Theseus were in when contact was first made, followed by the value of k.

Input will be from a file called PROBLEME.DAT and will consist of a series of lines. Each line will describe a scenario in the format shown below (which describes the above example). No line will contain more than 255 characters. The file will be terminated by a line consisting of a single #.

Output will consist of one line for each Labyrinth. Each line will identify the lit caverns, in the order in which the candles were left, and the cavern in which the Minotaur was trapped, following the format shown in the example below.

Example

INPUT
```
A:BCD;B:AD;D:BG;F:H;G:DEH;E:FGH;H:EG;C:AD.  A C 3
#
```

OUTPUT
```
D F G /E
```

Division I

Problem F — Xenosemantics

Contact with extra-terrestrial intelligence has been made at last!! A stream of messages has been discovered, apparently emanating from Procyon IV. After intensive study by the world's best xenosemanticists, the following definite conclusions on the format of the messages have been reached. The messages are streams of bits divided into groups of 8. Somewhat coincidentally the meaningful parts of the message map onto the lower case alphabet, although other characters sometimes intervene. Letters are organised into words separated by spacer letters. The spacer letter varies within a message, but a word which is delimited by a particular spacer pair does not contain that spacer letter within it. In addition the message is conceptually bounded by a pair of 'joker' letters or 'wild spacers' that can match any letter. For example, a message segment *xwrxwtx* contains 3 words - *wr, wt,* and *rx*; *wrxwt* is not a word in this segment of the message. If this segment appeared at the start of a message then *xw* and *xwrxw* could also be words. The words *wr* and *rx* overlap, while *wt* does not overlap any words in this message segment. While a word contains the same letters each time it appears in one message, the order of the letters may vary in different occurrences of the same word. Each message contains many words which are not "true" words in that they carry no meaning (like err.., umm.., etc in English). Every true word in the message contains at least two and no more than 250 letters, overlaps with another true word, and is repeated somewhere in the message (possibly with the letters in a different order). In the example above, *wr* and *rx* would both be true words if *wr* or *rw*, and *rx* or *xr*, occurred as words elsewhere in the message. The word *wt* would be a true word if *wt* or *tw* occurred elsewhere in the message, overlapping another true word.

Write a program that will read in messages and print out a list of the different true words contained in each message (using the spelling which occurs first), in the order the words first appear in the message. If the first appearances of two words overlap, then the word that finishes first precedes the other. Remember that both the start and the end of the message count as spacer letters. Your program must be able to process messages of up to 1000 letters.

Input will be from a file called PROBLEMF.DAT and will consist of one or more messages. Each message will consist of one or more lines. Each line will be no more than 60 characters long and will contain a mixture of lower case letters and other characters. If the last character of a line is a dash (-) then the message continues on the next line. All characters other than lower case 'a' to 'z' form no part of the message. The file will be terminated by a line consisting of a single #.

Output will consist of the true words for each message, in the correct order as specified above, one word per line. Terminate the list for each message by a line consisting of a single *.

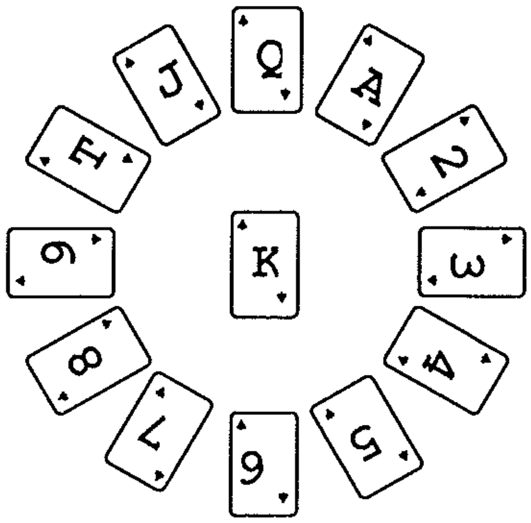
Example

INPUT
```
dyj@ttdi%sdort^jdyt*rFnn  trlnsvkGHoalexotrjxzasvs-
ozgpsi<>:pkelaovo,.;'slnxt'][-prsjlntrjo
aaaaaaa
#
```

OUTPUT
```
dyj
ortj
lnsvkoalexot
*
*
```

Problem G Clock Patience

Card sharp Albert (Foxy) Smith is writing a book on patience games. To double check the examples in the book, he is writing programs to find the optimal play of a given deal. The description of Clock Patience reads as follows: "The cards are dealt out (face down) in a circle, representing a clock, with a pile in each hour position and an extra pile in the centre of the clock. The first card goes face down on one o'clock, the next on two, and so on clockwise from there, with each thirteenth card going to the center of the clock. This results in thirteen piles, with four cards face down in each pile.



The game then starts. The top card of the 'king' pile (the last card dealt) is exposed to become the current card. Each move thereafter consists of placing the current card face up beneath the pile corresponding to its value and exposing the top card of that pile as the new current card. Thus if the current card is an Ace it is placed under the 'one' pile and the top card of that pile becomes the current card. The game ends when the pile indicated by the current card has no face down cards in it. You win if the entire deck is played out, i.e. exposed."

Write a program that will read in a number of shuffled decks, and play the game. The input will be from a file called PROBLEMG.DAT, and will consist of decks of cards arranged in four lines of 13 cards, cards separated by a single blank. Each card is represented by two characters, the first is the rank (A, 2, 3, 4, 5, 6, 7, 8, 9, T, J, Q, K) followed by the suit (H, D, C, S). The input will be terminated by a line consisting of a single #. The deck is listed from bottom to top, so the first card dealt is the last card listed.

The output will consist of one line per deck. Each line will contain the number of cards exposed during the game (2 digits, with a leading zero if necessary), a comma, and the last card exposed (in the format used in the input).

Example

INPUT

```
TS QC 8S 8D QH 2D 3H KH 9H 2H TH KS KC
9D JH 7H JD 2S QS TD 2C 4H 5H AD 4D 5D
6D 4S 9S 5S 7S JS 8H 3D 8C 3S 4C 6S 9C
AS 7C AH 6H KD JC 7D AC 5C TC QD 6C 3C
#
```

OUTPUT

```
44,KD
```

Division I

Problem H — Car Trialling

Car trialling requires the following of carefully worded instructions. When setting a trial, the organiser places traps in the instructions to catch out the unwary.

Write a program to determine whether an instruction obeys the following rules, which are loosely based on real car trialling instructions. **BOLD-TEXT** indicates text as it appears in the instruction (case sensitive), | separates options of which exactly one must be chosen, and .. expands, so **A..D** is equivalent to **A|B|C|D** .

instruction = navigational | time-keeping | navigational **AND** time-keeping
navigational = directional | navigational **AND THEN** directional
directional = how direction | how direction where
how = **GO** | **GO** when | **KEEP**
direction = **RIGHT** | **LEFT**
when = **FIRST** | **SECOND** | **THIRD**
where = **AT** sign
sign = "signwords"
signwords = s-word | signwords s-word
s-word = letter | s-word letter
letter = **A..Z** | **.**
time-keeping = record | change
record = **RECORD TIME**
change = cas **TO** nnn **KMH**
cas = **CHANGE AVERAGE SPEED** | **CAS**
nnn = digit | nnn digit
digit = **0..9**

Note that s-word and nnn are sequences of letters and digits respectively, with no intervening spaces. There will be one or more spaces between items except before a period (.), after the opening speech marks or before the closing speech marks.

The input will be from a file called PROBLEMH.DAT. Each line will consist of not more than 75 characters. The input will be terminated by a line consisting of a single #.

The output will consist of a series of sequentially numbered lines, either containing the valid instruction, or the text "Trap!" if the line did not obey the rules. The line number will be right justified in a field of 3 characters, followed by a full-stop, a single space, and the instruction, with sequences of more than one space reduced to single spaces.

Example:

INPUT
```
KEEP LEFT AND THEN GO RIGHT
CAS TO 20 KMH
GO FIRST       RIGHT AT "SMITH ST."  AND   CAS TO 20 KMH
GO 2nd RIGHT
GO LEFT AT "SMITH STREET AND RECORD TIME
KEEP RIGHT AND THEN RECORD TIME
#
```

OUTPUT
```
  1. KEEP LEFT AND THEN GO RIGHT
  2. CAS TO 20 KMH
  3. GO FIRST RIGHT AT "SMITH ST." AND CAS TO 20 KMH
  4. Trap!
  5. Trap!
  6. Trap!
```