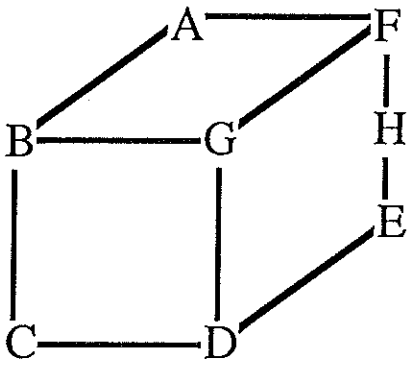
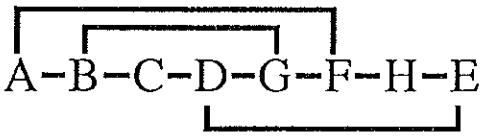
Problem A Bandwidth

Given a graph (V,E) where V is a set of nodes and E is a set of arcs in VxV, and an *ordering* on the elements in V, then the *bandwidth* of a node *v* is defined as the maximum distance in the ordering between *v* and any node to which it is connected in the graph. The bandwidth of the ordering is then defined as the maximum of the individual bandwidths. For example, consider the following graph:



This can be ordered in many ways, two of which are illustrated below:



For these orderings, the bandwidths of the nodes (in order) are 6, 6, 1, 4, 1, 1, 6, 6 giving an ordering bandwidth of 6, and 5, 3, 1, 4, 3, 5, 1, 4 giving an ordering bandwidth of 5.

Write a program that will find the ordering of a graph that minimises the bandwidth.

Input will be from a file called PROBLEMA.DAT, and will consist of a series of graphs. Each graph will appear on a line by itself. The entire file will be terminated by a line consisting of a single #. For each graph, the input will consist of a series of records separated by ';'. Each record will consist of a node name (a single upper case character in the the range 'A' to 'Z'), followed by a ':' and at least one of its neighbours. The graph will contain no more than 8 nodes.

Output will consist of one line for each graph, listing the ordering of the nodes followed by an arrow (->) and the bandwidth for that ordering. All items must be separated from their neighbours by exactly one space. If more than one ordering produces the same bandwidth, then choose the smallest in lexicographic ordering, i.e. the one that would appear first in an alphabetic listing.

Example

INPUT

```
A:FB;B:GC;D:GC;F:AGH;E:HD
#
```

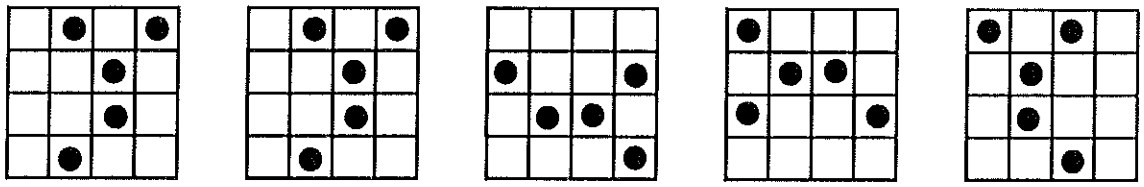
OUTPUT

```
A B C F G D H E -> 3
```

Problem B The Spot Game

The game of Spot is played on an NxN board as shown below for N = 4. During the game, alternate players may either place a black counter (spot) in an empty square or remove one from the board, thus producing a variety of patterns. If a board pattern (or its rotation by 90° or 180°) is repeated during a game, the player producing that pattern loses and the other player wins. The game terminates in a draw after 2N moves if no duplicate pattern is produced before then.

Consider the following patterns:



If the first pattern had been produced earlier, then any of the following three patterns (plus one other not shown) would terminate the game, whereas the last one would not.

Input will be from a file called PROBLEMB.DAT, and will consist of a series of games, each consisting of the size of the board, N ($2 \le N \le 50$) followed, on separate lines, by 2N moves, whether they are all necessary or not. Each move will consist of the coordinates of a square (integers in the range 1..N) followed by a blank and a character '+' or '-' indicating the addition or removal of a spot respectively. You may assume that all moves are legal, i.e. there will never be an attempt to place a spot on an occupied square, nor to remove a non-existent spot. The entire file will be terminated by a zero (0).

Output will consist of one line for each game indicating which player won and on which move, or that the game ended in a draw.

Example

INPUT

```
2
1 1 +
2 2 +
2 2 -
1 2 +
2
1 1 +
2 2 +
1 2 +
2 2 -
0
```

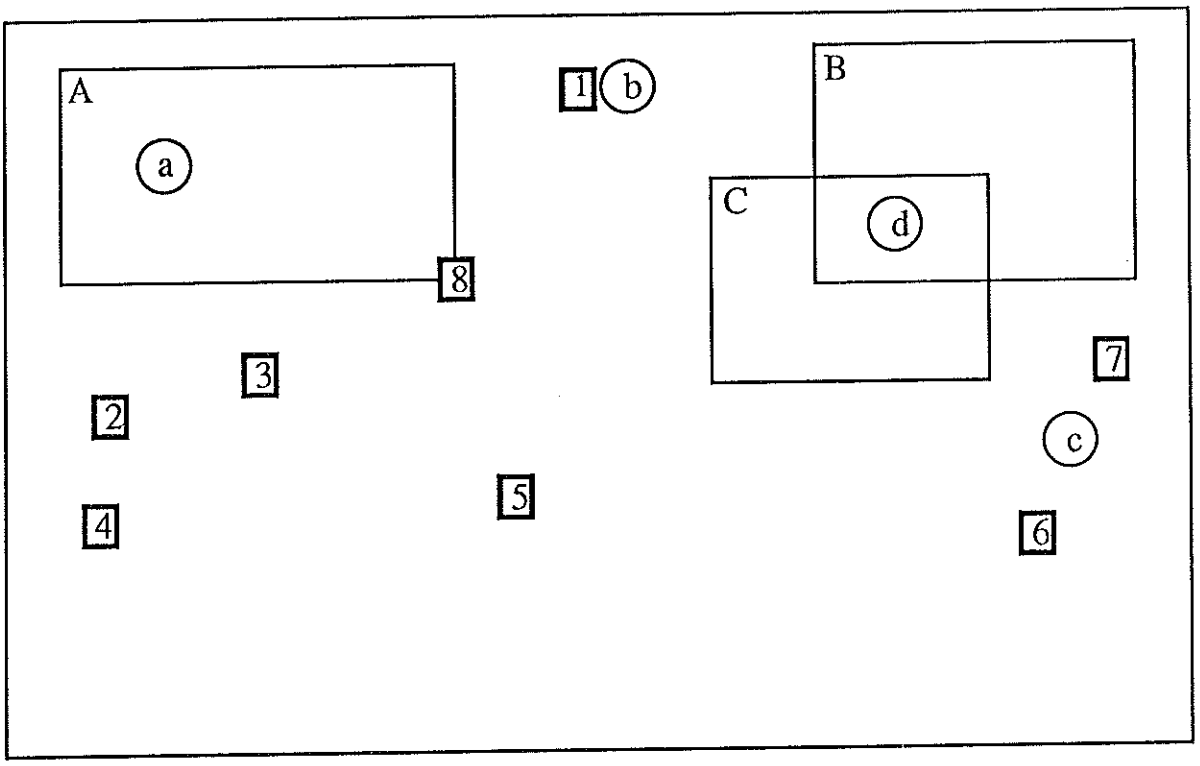
OUTPUT

```
Player 2 wins on move 3
Draw
```

Problem C Mouse Clicks

A typical windowing system on a computer will provide a number of icons on the screen as well as some defined regions. When the mouse button is clicked, the system has to determine where the cursor is and what is being selected. For this problem we assume that a mouse click in (or on the border of) a region selects that region, otherwise it selects the closest visible icon (or icons in the case of a tie).

Consider the following screen:



A mouse click at 'a' will select region A. A mouse click at 'b' will select icon 1. A mouse click at 'c' will select icons 6 and 7. A mouse click at 'd' is ambiguous. The ambiguity is resolved by assuming that one region is in front of another. In the data files, later regions can be assumed to be in front of earlier regions. Since regions are labelled in order of appearance (see later) 'd' will select C. Note that regions always overlap icons so that obscured icons need not be considered and that the origin (0,0) is at the top left corner.

Write a program that will read in a file of region and icon definitions followed by a series of mouse clicks and return the selected items. Coordinates will be given as pairs of integers in the range 0..499 and you can assume that all icons and regions lie wholly within the screen. Your program must number all icons (even invisible ones) in the order of arrival starting from 1 and label regions alphabetically in the order of arrival starting from 'A'.

Input will be from a file called PROBLEMC.DAT, and will consist of a series of lines. Each line will identify the type of data: I for icon, R for region and M for mouse click. There will be no separation between the specification part and the event part, however no icon or region specifications will follow the first mouse click. An I will be followed by the coordinates of the centre of the icon, R will be followed by the coordinates of the top left andf bottom right corners respectively and M will be followed by the coordinates of the cursor at the time of the click. There will always be at least one visible icon and never more than 25 regions and 50 icons. The entire file will be terminated by a line consisting of a single #.

Output will consist of one line for each mouse click, containing the selection(s) for that click. Regions will be identified by their single character identifier, icon numbers will be written out right justified in a field of width 3.

Example

INPUT

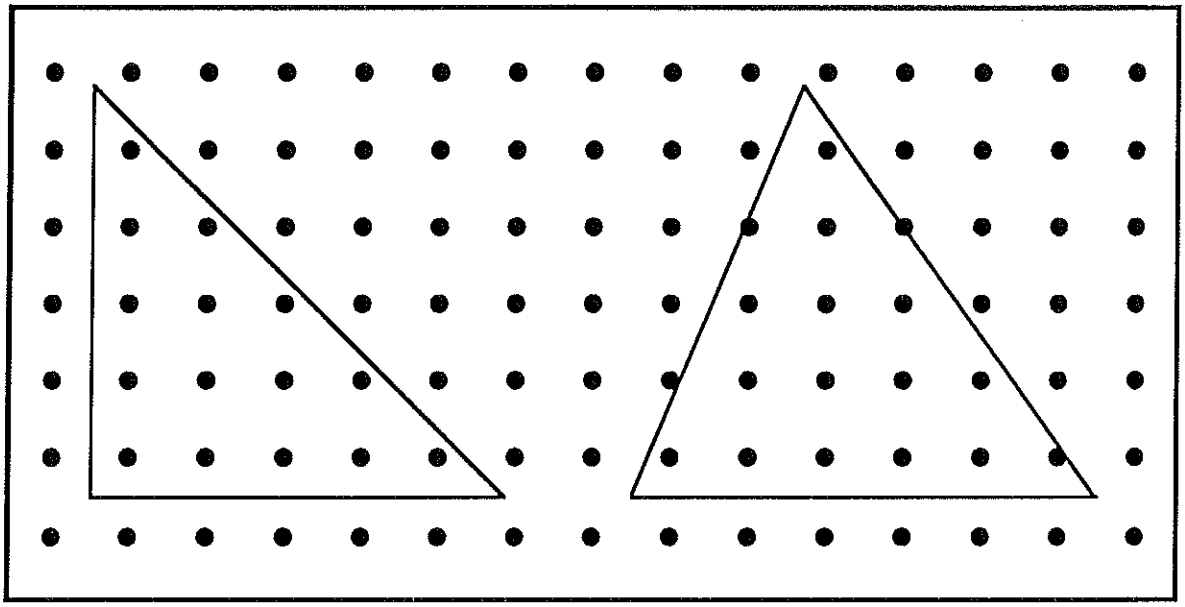
```
I  216    28
R   22    19   170   102
I   40   150
I   96   138
I   36   193
R  305    13   425   103
I  191   184
I  387   200
R  266    63   370   140
I  419   134
I  170   102
M   50    50
M  236    30
M  403   167
M  330    83
#
```

OUTPUT

```
A
  1
  6  7
C
```

Problem D Orchard Trees

An Orchardist has planted an orchard in a rectangle with trees uniformly spaced in both directions. Thus the trees form a rectangular grid and we can consider the trees to have integer coordinates. The origin of the coordinate system is at the bottom left of the following diagram:



Consider that we now overlay a series of triangles on to this grid. The vertices of the triangle can have any real coordinates in the range 0.0 to 100.0, thus trees can have coordinates in the range 1 to 99. Two possible triangles are shown.

Write a program that will determine how many trees are contained within a given triangle. For the purposes of this problem, you may assume that the trees are of point size, and that any tree (point) lying exactly on the border of a triangle is considered to be in the triangle.

Input will be from a file called PROBLEMD.DAT, and will consist of a series of lines. Each line will contain 6 real numbers in the range 0.00 to 100.00 representing the coordinates of a triangle. The entire file will be terminated by a line containing 6 zeroes (0 0 0 0 0 0).

Output will consist of one line for each triangle, containing the number of trees for that triangle right justified in a field of width 4.

Example

INPUT

```
1.5 1.5  1.5 6.8  6.8 1.5
10.7 6.9  8.5 1.5  14.5 1.5
```

OUTPUT

```
  15
  17
```

Problem E Student Grants

The Government of Impecunia has decided to discourage tertiary students by making the payments of tertiary grants a long and time-consuming process. Each student is issued a student ID card which has a magnetically encoded strip on the back which records the payment of the student grant. This is initially set to zero. The grant has been set at $40 per year and is paid to the student on the working day nearest to his birthday. (Impecunian society is still somewhat medieval and only males continue with tertiary education.) Thus on any given working day up to 25 students will appear at the nearest office of the Department of Student Subsidies to collect their grant.

The grant is paid by an Automatic Teller Machine which is driven by a reprogrammed 8085½ chip originally designed to run the state slot machine. The ATM was built in the State Workshops and is designed to be difficult to rob. It consists of an interior vault where it holds a large stock of $1 coins and an output store from which these coins are dispensed. To limit possible losses it will only move coins from the vault to the output store when that is empty. When the machine is switched on in the morning, with an empty output store, it immediately moves 1 coin into the output store. When that has been dispensed it will then move 2 coins, then 3, and so on until it reaches some preset limit k. It then recycles back to 1, then 2 and so on.

The students form a queue at this machine and, in turn, each student inserts his card. The machine dispenses what it has in its output store and updates the amount paid to that student by writing the new total on the card. If the student has not received his full grant, he removes his card and rejoins the queue at the end. If the amount in the store plus what the student has already received is more than $40, the machine only pays out enough to make the total up to $40. Since this fact is recorded on the card, it is pointless for the student to continue queuing and he leaves. The amount remaining in the store is then available for the next student.

Write a program that will read in values of N (the number of students, $1 \le N \le 25$) and k (the limit for that machine, $1 \le k \le 40$) and calculate the order in which the students leave the queue.

Input will be from a file called PROBLEME.DAT. It will consist of a series of lines each containing a value for N and k as integers. The list will be terminated by two zeroes (0 0).

Output will consist of a line for each line of input and will contain the list of students in the order in which they leave the queue. Students are ordered according to their position in the queue at the start of the day. All numbers must be right justified in a field of width 3.

Example

INPUT

```
5 3
0 0
```

OUTPUT:

```
  1  3  5  2  4
```

Problem F Gondwanaland Telecom

Gondwanaland Telecom makes charges for calls according to distance and time of day. The basis of the charging is contained in the following schedule, where the charging step is related to the distance:

Charging Step (distance)	Day Rate 8am to 6pm	Evening Rate 6pm to 10pm	Night Rate 10pm to 8am
A	0.10	0.06	0.02
B	0.25	0.15	0.05
C	0.53	0.33	0.13
D	0.87	0.47	0.17
E	1.44	0.80	0.30

All charges are in dollars per minute of the call. Calls which straddle a rate boundary are charged according to the time spent in each section. Thus a call starting at 5:58 pm and terminating at 6:04 pm will be charged for 2 minutes at the day rate and for 4 minutes at the evening rate. Calls less than a minute are not recorded and no call may last more than 24 hours.

Write a program that reads call details and calculates the corresponding charges.

Input will be from a file called PROBLEMF.DAT. Each line of data will consist of the charging step (upper case letter 'A' .. 'E'), the number called (a string of 7 digits and a hyphen in the approved format) and the start and end times of the call, all separated by exactly one blank. Times are recorded as hours and minutes in the 24 hour clock, separated by one blank and with two digits for each number. Input will be terminated by a line consisting of a single #.

Output will consist of the called number, the time the call spent in each of the charge categories, the charging step and the total cost in the format shown below.

Example

INPUT

```
A 183-5724 17 58 18 04
#
```

OUTPUT

```
         10        16      22      28  31      39
  183-5724         2       4       0   A    0.44
```

Problem G ID Codes

It is 2084 and the year of Big Brother has finally arrived, albeit a century late. In order to exercise greater control over its citizens and thereby to counter a chronic breakdown in law and order, the Government decides on a radical measure — all citizens are to have a tiny microcomputer surgically implanted in their left wrists. This computer will contains all sorts of personal information as well as a transmitter which will allow people's movements to be logged and monitored by a central computer. (A desirable side effect of this process is that it will shorten the dole queue for plastic surgeons.)

An essential component of each computer will be a unique identification code, consisting of up to 50 characters drawn from the 26 lower case letters. The set of characters for any given code is chosen somewhat haphazardly. The complicated way in which the code is imprinted into the chip makes it much easier for the manufacturer to produce codes which are rearrangements of other codes than to produce new codes with a different selection of letters. Thus, once a set of letters has been chosen all possible codes derivable from it are used before changing the set.

For example, suppose it is decided that a code will contain exactly 3 occurrences of 'a', 2 of 'b and 1 of 'c', then three of the allowable 60 codes under these conditions are:

abaabc
abaacb
ababac

These three codes are listed from top to bottom in alphabetic order. Among all codes generated with this set of characters, these codes appear consecutively in this order.

Write a program to assist in the issuing of these identification codes. Your program will accept a sequence of no more than 50 lower case letters (which may contain repeated characters) and print the successor code if one exists or the message 'No Successor' if the given code is the last in the sequence for that set of characters.

Input will be from a file called PROBLEMG.DAT, and will consist of a series of lines each containing a string representing a code. The entire file will be terminated by a line consisting of a single #.

Output will consist of one line for each code read containing the successor code or the words 'No Successor'.

Example

INPUT

```
abaacb
cbbaa
#
```

OUTPUT

```
ababac
No Successor
```

Problem H Dollars

New Zealand currency consists of $100, $50, $20, $10, and $5 notes and $2, $1, 50c, 20c, 10c and 5c coins. Write a program that will determine, for any given amount, in how many ways that amount may be made up. Changing the order of listing does not increase the count. Thus 20c may be made up in 4 ways: 1x20c, 2x10c, 10c+2x5c, and 4x5c.

Input will be from a file called PROBLEMH.DAT, and will consist of a series of real numbers no greater than $50.00 each on a separate line. Each amount will be valid, i.e. will be a multiple of 5c. The file will be terminated by a line containing zero (0.00).

Output will consist of a line for each of the amounts in the input, each line consisting of the amount of money (with two decimal places and right justified in a field of width 5), followed by the number of ways in which that amount may be made up, right justified in a field of 12 places.

Example

INPUT

```
0.20
  2.00
0.00
```

OUTPUT

```
    5                  17
 0.20                   4
 2.00                 293
```

Problem I

π

One way of determining π is to use the arctan power series :

$$\arctan(x) = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \ldots$$

together with the fact that $\pi/4 = \arctan(1)$. This method is improved considerably by trigonometric relationships like :

$$\pi/4 = 4\arctan(1/5) - \arctan(1/239)$$

Use these two formulae to find π to 1,000,000 decimal places.

Problem P Scrambling Words

A very simple code, which has been used successfully for ages, is to simply scramble the words in a document. You may think that this would be easy to decode (surely 'giD rednu eht esor hsub' can't hide its secret for too long), but this assumes you know what language it is written in. Choosing an unfamiliar language, and scrambling the words, is quite a powerful encoding technique. But with the advent of computers, it is easy to automatically produce all possible unscramblings, and leave it to a human being to sort out what the correct text is.

Write a program which will read in a line of text, and produce lines which correspond to re-arrangements of the letters in each word. The first line of your output should give the text, with the letters in each word reversed (so 'buttons' becomes 'snottub'). The next line should give the text, with all adjacent pairs of letters swapped (so 'buttons' becomes 'ubttnos' - any letters left over are left unchanged). The next five lines contains the text, with all possible rearrangements of three adjacent letters (so 'buttons' becomes 'ubtotns', then 'tubnots', then 'btutnos', then 'tbuntos', then 'utbonts').

Input will be a single line typed in from the keyboard. Words will be separated by one or more spaces. Output will be seven lines, with the letters in each word rearranged as described above, and the words separated by the same number of blanks as the original text.

Example
INPUT
giD rednu eht esor hsub

OUTPUT
Dig under the rose bush
igD erndu het sero shbu
igD erdnu het seor shub
Dig dernu the oser ushb
gDi rdenu eth eosr husb
Dgi drenu teh oesr uhsb
iDg edrnu hte soer suhb

Problem Q Ramanujan Numbers

Ramanujan was an Indian mathematician who lived in the beginning of this century. He failed the English examination for University Entrance, and was too poor to buy mathematics books, but he worked it all out by himself, and started producing important mathematical theorems of his own! He was particularly interested in number theory, and it was said that 'every positive integer was his personal friend'. There is a story about the mathematician Hardy who went to see him once, and who commented that the number of the cab he had travelled in was an uninteresting number, 1729. "On the contrary", replied Ramanujan, "that is a most interesting number; it is the smallest number that can be expressed as the sum of two cubes in two different ways"

$$1729 = 12^3 + 1^3 = 10^3 + 9^3$$

Write a program which will calculate all the numbers less than 1,000,000 which have this property (they can be expressed as the sum of the cubes of two positive integers in two different ways). There are 43 of them. Your program must print them out in increasing order (the first is 1729) and must take less than 2 minutes to run.

There is no input. Output is a list of 43 numbers in increasing order, one on each line, right justified in a field of width 7.

Problem R Writing Numbers

When a cheque is written, the amount must be expressed in words as well as digits.to make it valid. This is a problem for computers, and cheques are usually printed on special stationary which makes this task easy. Unfortunately, if inflation reamains high, we will shortly be writing cheques for billions, trillions etc of dollars, and this method will no longer work.

Write a program which will produce the 'word' version of any number which is given to it (up to 21 digits long), correctly formatted. For example, 132450618100000041101 is written as :

```
one hundred and thirty two quintillion four hundred and fifty
quadrillion six hundred and eighteen trillion one hundred billion
forty one thousand one hundred and one.
```

Input will be a single string of up to 21 digits typed in from the keyboard. Output will be lines giving the number in words correctly formatted. Make sure no more than seventy characters are on each line, and no word is broken across two lines.

Example
INPUT
111001200

OUTPUT
one hundred and eleven million one thousand two hundred

Problem S Runs and Flushes

Write a program that will read in a representation of a deck of cards and then determine the longest flush and the longest run present in the deck. A *flush* is a series of cards all of the same suit and a *run* is a series of cards that follow one another in *ascending* numerical sequence, with Ace following King and preceding Two. Thus ♠2 ♠5 ♠K ♠3 ♠A forms a flush of length 5, ♣9 ♦10 ♣J ♠Q ♥K ♣A ♦2 forms a run of length 7 and ♥2 ♥3 ♥4 ♥5 ♥6 forms both a flush and a run of length 5.

Input will be from a file called PROBLEMS.DAT, and will consist of a series of decks of cards, each deck occupying two lines. The input will be terminated by a line consisting of a single #. Each card will be represented by a two character string — the suit (S, H, D, C) and the value (A for Ace, 2 – 9 for two through nine,T for ten, J for Jack, Q for Queen and K for King). There will be 26 cards per line.

Output will consist of one line for each deck and will contain the length of the longest flush followed by the length of the longest run, each right justified in a field of width 6.

Example

INPUT

```
CQDTC4D8S7HTDAH7D2S3D6C6S6D9S4SAD7H2CKH5D3CTS8C9H3C3
DQS9SQDJH8HAS2SKD4H4S5C7SJC8DKC5C2CAHQCJSTH6HKH9D5HJ
#
```

OUTPUT

```
     6    12
     3     2
```

Problem T Family Concepts

An interesting parlour game is known as 'Family Concepts'. One person thinks of a concept that defines a class of word and then describes a family as liking instances of the concept and not liking non-instances of the concept. When a member of the party guesses the concept, she can assist in describing this mythical family. As more and more participants guess the concept they too enter the game, and the descriptions usually get more and more outrageous, until every one has guessed the concept and joined in the fun.

A typical concept is 'the word contains a double letter' in which case the family would:

like sheep and cattle but neither pigs nor horses,
eat sweets but not chocolates,
live in a village, but not in a town or city
etc.

Write a program that will determine whether a given word fits this concept, i.e. has a double letter, or not.

Input will be from a file called PROBLEMT.DAT, and will consist of a series of lines, each containing a word consisting of up to 20 lower case letters only. The file will be terminated by a line consisting of a single #.

Output will consist of a series of lines, one for each line in the input file. Each line will consist of the word starting in column 1 and then, ending in column 24, one of the words 'yes' or 'no' depending on whether the word has at least two adjacent letters the same.

Example

INPUT

```
street
village
chocolates
abcdefghijklmnopqrst
#
```

OUTPUT

```
1                        24
street                  yes
village                 yes
chocolates               no
abcdefghijklmnopqrst     no
```

Problem U — Russell Soundex Coding

In many situations it is desirable that surnames that sound similar should occur together, for instance all the variants of Smith (Smit, Smythe) should appear together in a telephone directory. One system of encoding names to assist in this is known as the Russell Soundex encoding.

The rules for encoding are as follows:

1. The code consists of the initial letter of the surname followed by 3 digits derived from subsequent letters in the name.
2. The letters A, E, H, I, O, U, W, Y are pseudo-vowels and do not form part of the code - the remaining letters are called codable letters. Hyphens, quotes, etc are ignored.
3. Initially all codable letters in the surname (including the first) are encoded according to the following table, although they may not all be used.

1	B, F, P, V
2	C, G, J, K, S, X, Z
3	D, T
4	L
5	M, N
6	R

4. The encoding is then scanned for sequences of repeated digits. All but the first of these are discarded unless the letters they were derived from are separated by a pseudo-vowel.
5. The first three remaining digits are used as the code. If there are less than three digits, the code is padded with zeroes to the right.

Some examples follow to illustrate the workings of the various rules.

```
IRVINE     =  I R V I N E
              I 6 1 * 5 * = I615        coding
              1 3 3 2 3 2               rules used

SMITHSON   =  S M I T H S O N
              S 5 * 3 * 2 * * = S532    coding
              1 3 2 3 2 3 2 5           rules used

LLEWELLYN  =  L L E W E L L Y N
              L * * * * 4 * * 5 = L450  coding
              1 4 2 2 2 3 4 2 2 (5)     rules used
```

Write a program that will accept surnames and determine their codes.

Input will be from a file called PROBLEMU.DAT and will consist of a series of surnames, one per line. Each surname will be a string containing no more than 20 characters. All letters will be in upper case. The input will be terminated by a line consisting of a single #.

Output will consist of a series of lines containing the given surname and the encoding for that name, in the format shown below.

Example

INPUT

```
IRVINE
LLEWELLYN
#
```

OUTPUT

```
1                    22
IRVINE               I615
LLEWELLYN            L450
```

Problem V Caesar Cypher

One of the earliest encrypting systems is attributed to Julius Caesar: if the letter to be encrypted is the Nth letter in the alphabet, replace it with the (N+K)th where K is some fixed integer (Caesar used K = 3). We usually treat a space as zero and all arithmetic is then done modulo 27. Thus for K = 1 the message 'ATTACK AT DAWN' becomes 'BUUBDLABUAEBXO'.

Decrypting such a message is trivial since one only needs to try 26 different values of K. This process is aided by knowledge of the language, since then one can determine when the decrypted text forms recognisable words. If one does not know the language, then a dictionary would be necessary.

Write a program that will read in a dictionary and some encrypted text, determine the value of K that was used, and then decrypt the cyphertext to produce the original message. The original message contained only letters and spaces and has been encrypted using the above method. The most suitable value of K will be the one which produces the most matches with the words in the dictionary.

Input will be from a file called PROBLEMV.DAT. It will consist of a dictionary and the encrypted text. The dictionary will consist of no more than 100 lines each containing a word in upper case characters and not more than 20 characters in length. The dictionary portion will be terminated by a line consisting of a single #. The encrypted text will follow immediately and will consist of a single line containing no more than 250 characters. Note that the dictionary will not necessarily contain all the words in the original text, although it will certainly contain a large proportion of them. It may also contain words that are not in the original text. The dictionary will not appear in any particular order.

Output will consist of the decrypted text. Lines should be as long as possible, but not exceeding 60 characters and no word may cross a line break.

Example

INPUT

```
THIS
DAWN
THAT
THE
ZORRO
OTHER
AT
THING
#
BUUBDLA PSSPABUAEBXO
```

OUTPUT

```
ATTACK ZORRO AT DAWN
```

Problem W Gondwanaland Telecom

Gondwanaland Telecom makes charges for calls according to distance and time of day. The basis of the charging is contained in the following schedule, where the charging step is related to the distance:

Charging Step (distance)	Day Rate 8am to 6pm	Evening Rate 6pm to 10pm	Night Rate 10pm to 8am
A	0.10	0.06	0.02
B	0.25	0.15	0.05
C	0.53	0.33	0.13
D	0.87	0.47	0.17
E	1.44	0.80	0.30

All charges are in dollars per minute of the call. Calls which straddle a rate boundary are charged according to the time spent in each section. Thus a call starting at 5:58 pm and terminating at 6:04 pm will be charged for 2 minutes at the day rate and for 4 minutes at the evening rate. Calls less than a minute are not recorded and no call may last more than 24 hours.

Write a program that reads call details and calculates the corresponding charges.

Input will be from a file called PROBLEMW.DAT. Each line of data will consist of the charging step (upper case letter 'A' .. 'E'), the number called (a string of 7 digits and a hyphen in the approved format) and the start and end times of the call, all separated by exactly one blank. Times are recorded as hours and minutes in the 24 hour clock, separated by exactly one space, with two digits for each number. Input will be terminated by a line consisting of a single #.

Output will be to the screen, although it may be redirected to a file or piped to another program. It will consist of the called number, the time the call spent in each of the charge categories, the charging step and the total cost in the format shown below.

Example

INPUT

```
A 183-5724 17 58 18 04
#
```

OUTPUT

```
          10        16        22        28  31        39
  183-5724         2         4         0   A      0.44
```

Problem X Student Grants

The Government of Impecunia has decided to discourage tertiary students by making the payments of tertiary grants a long and time-consuming process. Each student is issued a student ID card which has a magnetically encoded strip on the back which records the payment of the student grant. This is initially set to zero. The grant has been set at $40 per year and is paid to the student on the working day nearest to his birthday. (Impecunian society is still somewhat medieval and only males continue with tertiary education.) Thus on any given working day up to 25 students will appear at the nearest office of the Department of Student Subsidies to collect their grant.

The grant is paid by an Automatic Teller Machine which is driven by a reprogrammed 8085½ chip originally designed to run the state slot machine. The ATM was built in the State Workshops and is designed to be difficult to rob. It consists of an interior vault where it holds a large stock of $1 coins and an output store from which these coins are dispensed. To limit possible losses it will only move coins from the vault to the output store when that is empty. When the machine is switched on in the morning, with an empty output store, it immediately moves 1 coin into the output store. When that has been dispensed it will then move 2 coins, then 3, and so on until it reaches some preset limit k. It then recycles back to 1, then 2 and so on.

The students form a queue at this machine and, in turn, each student inserts his card. The machine dispenses what it has in its output store and updates the amount paid to that student by writing the new total on the card. If the student has not received his full amount, he removes his card and rejoins the queue at the back. If the amount in the store plus what the student has already received comes to more than $40, the machine only pays out enough to make the total up to $40. Since this fact is recorded on the card, it is pointless for the student to continue queuing and he leaves. The amount remaining in the store is then available for the next student.

Write a program that will read in values of N (the number of students, $1 \le N \le 25$) and k (the limit for that machine, $1 \le k \le 40$) and calculate the order in which the students leave the queue.

Input will be from a file called PROBLEMX.DAT. It will consist of a series of lines each containing a value for N and k as integers. The list will be terminated by two zeroes (0 0).

Output will consist of a single line for each situation and will contain the list of students in the order in which they leave the queue. Students are ordered according to their position in the queue at the start of the day. All numbers must be right justified in a field of width 3.

Example

INPUT

```
5, 3
0 0
```

OUTPUT:

```
  1  3  5  2  4
```

Problem Y Amicable Numbers

The factor sum of a number is the sum of all its factors, including 1 but excluding itself. Thus the factor sum of 8 is 1 + 2 + 4 = 7, the factor sum of 25 is 1 + 5 = 6. (Note that a factor of a number divides it exactly).

We define amicable pairs as numbers where each member of the pair is the factor sum of the other, e.g. 284 and 220.

> Factor sum of 284 is 1 + 2 + 4 + 71 + 142 = 220
> Factor sum of 220 is 1 + 2 + 4 + 5 + 10 + 11 + 20 + 22 + 44 + 55 + 110 = 284

Write a program that will read in pairs of numbers and determine whether or not they form an amicable pair.

Input will be from a file called PROBLEMY.DAT. Each line of data will consist of a pair of integers less than 10,000. Input will be terminated by a line containing a pair of zeroes (0 0).

Output will consist of a series of lines, one for each line of the input, containing the numbers right justified in a field of width 5 and, starting in column 12, one of the words 'yes' or 'no', depending on whether the numbers form an amicable pair or not.

Example

INPUT

```
284 220
1280 1786
0 0
```

OUTPUT

```
    5    10 12
  284   220 yes
 1280  1786 no
```

Problem Z String Transformations

Write a program that will accept strings of up to 60 characters and transform them according to the following rules.

Transform all uppercase letters to the equivalent lower case characters, similarly transform all lower case characters to uppercase characters. All blanks are to be transformed to '#' and all other characters become '?'.

Input will be from a file called PROBLEMZ.DAT. Each line of data will consist of a string to be transformed. Input will be terminated by a line consisting of a single #.

Output will consist of the transformed strings, one per line.

Example

INPUT

```
This, believe it (or not) is a String//!
SO IS THIS
#
```

OUTPUT

```
tHIS?#BELIEVE#IT#?OR#NOT?#IS#A#sTRING???
so#is#this
```