2011/2012 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 1
Change by Mass

Your team is working on software for automated check-out machines (such as are found in supermarkets and large home-improvement stores). The usual change-making process for these machines minimizes the number of coins dispensed when dispensing amounts under one dollar. However, a customer wants to change the logic in the machines to minimize the mass of the coins dispensed (to "lighten the load" for their customers). Minimizing the number of coins dispensed remains a secondary goal.

The masses of United States coins with values of less than one dollar are as follows:

Cent ("Penny")	2.500 g
Five Cents ("Nickel")	5.000 g
Dime	2.268 g
Quarter Dollar	5.670 g
Half Dollar	11.340 g

Your team is to write a program that will, given an amount in cents, determine the coins to be dispensed that will minimize the total mass of the dispensed coins, and secondarily minimize the number of coins dispensed. Half-dollar coins may or may not be available, as they do not circulate to the extent the other coins do. You may assume that the check-out machines will have adequate supplies of coins of the other denominations.

Input to your program will be a series of change requests, one per line. Each change request will consist of the amount to be dispensed (integer cents, in the range 1–99 inclusive) starting in the first column, followed by a single space and the number of half-dollar coins in the machine (as an integer ≥ 0). Input will be terminated by the end-of-file.

For each request, your program should produce a line containing the coins to be dispensed. The output line is to show the denominations being dispensed in decreasing order, in the format "nxd" (meaning n coins of denomination d are to be dispensed). For example, 35 cents would be dispensed as "`1x25 1x10`". Denomination entries are to be separated from each other by single spaces, and no leading or trailing whitespace is to appear on an output line.

Sample Input

```
35 1
1 0
18 3
60 0
```

Output for the Sample Input

```
1x25 1x10
1x1
1x10 1x5 3x1
2x25 1x10
```

Problem 2
Planet "difftime"

NASA has established communications with a new planet discovered by Dr. Ridgemont. This planet, coincidentally, has a calendar very similar to the calendar we use here on Earth. But the minutes, hours, days, and so forth are all of a different duration. For example, on Ridgemont's Planet, a minute might last 54 seconds, there might be 61 minutes per hour, 23 hours per day, and 340 days per year. The year is split into months, and as on Earth, each month may have a different number of days. The inhabitants have set the calendar this way so that there is no need to handle leap years or daylight savings time, but other than that everything is different.

NASA software engineers want to practice good code reuse, so naturally they want to port their Linux source code into the communications system they use to talk to the inhabitants on this distant world. Much of their software utilizes the function *difftime*, which takes two specifications of a date and time and returns the number of seconds between them. For example, on Earth, if I ask for the difference between 12:35:00 on one day and 12:35:15 on the next day, the answer is 86,415. This represents one days worth of seconds (24 hours times 60 minutes times 60 seconds is 86,400) plus an additional 15 seconds.

For example, suppose that Ridgemont's Planet has 13 months and the number of days in each month as shown in the table below. There are a total of 399 days in a year.

Month	1	2	3	4	5	6	7	8	9	10	11	12	13
Days	30	30	31	31	31	31	31	31	31	31	31	30	30

The planet has 23 hours per day, each hour has 53 minutes, and each minute has 61 seconds.

A NASA engineer wants to know the difference between 22:30:05 on 02/29/11 and 10:00:00 on 03/01/11. By applying the above values, the engineer can determine that the elapsed time is 108,087 "planet seconds."

Your team is to write a program that reads the characteristics of time on a distant planet and two date/time specifications, and then prints the difference between the two times in seconds. You are guaranteed that the second date/time specification is not chronologically before the first.

There will be an arbitrary number of cases to consider. The input for each case will begin on a new line and consist of a series of values that span one or more lines. Each series will contain the following data items in the order specified, separated by one or more spaces and/or newlines. The allowable range of values for each item is given in parentheses following the item.

- The number of seconds in a minute (1–100).
- The number of minutes in an hour (1–100).
- The number of hours in a day (1–100).
- The number of months in a year (1–99), followed by that many integers, each of which represents the number of days in the corresponding month (1–99).
- A start date/time specification in the format "*MM/DD/YY-HH:MM:SS*".
- An end date/time specification in the format "*MM/DD/YY-HH:MM:SS*".

Like on Earth, months and days are numbered starting with one, and years, hours, minutes, and seconds are numbered starting with zero. The series following the input for the last case will contain a single zero.

For each case, your program is to print the difference between the start and end date/time specifications in seconds. Each result is to be printed on a line by itself, starting in the first column, without any trailing whitespace.

Sample Input

```
61 53 23 13 30 30 31 31 31 31 31 31 31 31 31 30 30 02/29/11-22:30:05
03/01/11-10:00:00
60 60 24 12 31 28 31 30 31 30 31 31 30 31 30 31
11/12/11-13:30:00 11/12/11-18:30:00
0
```

Output for the Sample Input

```
108087
18000
```

2011/2012 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 3
Graph Clue

Your team is competing in the annual Swamp County Scavenger Hunt. One of the clues must be discovered from a file that contains representations of graphs: the clue is the number of unique graphs in that file.

What counts is the shape of the graph, not the labels of the vertices or the way it is drawn. For example, the graphs of figures 1 and 2 are the same shape and they are said to be isomorphic. There is a way to relabel the vertices in figure 1 and redraw the graph so it is identical with figure 2. The graph in figure 3, on the other hand, cannot be made the same as the graph in figure 1 in spite of having the same number of edges and vertices.

The graphs are simple graphs: they are undirected and have no vertex connected to itself or multiple edges between any two vertices.

A simple graph with n vertices can be represented by an adjacency matrix with $A_{ij} = 1$ if there is an edge between vertex i and j. These are undirected graphs and no vertex is connected to itself so $A_{ij} = A_{ji}$ and $A_{ii} = 0$. Thus the diagonal elements $A_{11}, \ldots, A_{nn}$ are all zero and the matrix elements below the diagonal are computable from those above it.

The adjancency matrix for the graph in figure 1 is:

```
010110
101001
010110
101001
101001
010110
```

The input representation for a graph is the part of the adjacency matrix above the diagonal from left to right starting at the top row. Thus the input for the graph in figure 1 is:

```
101101001110011
```

Input to your program will be a series of lines ended by end-of-file. Each line is a graph representation as described above. There will be at most 100 lines and each graph will have between 2 and 8 vertices.

Print the number of unique graphs on one line with no leading zeroes or spaces and no trailing spaces.

Sample Input

```
101101001110011
001110111111000
111111
111001010001111
```

Output for the Sample Input

```
3
```

2011/2012 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 4
Bond Dutch Auctions

A bond is a debt instrument issued by a corporation or public entity to borrow money. Instead of (or in addition to) borrowing from banks, a corporation can issue bonds and sell them to investors. Bonds are issued in units of $1,000 and mature after a stated period of time (usually measured in years). A bond carries a given annual interest rate. Every six months, simple interest is paid on the bond at the stated rate, and at maturity the $1,000 principal is repaid as well. The interest payments are referred to as "coupon payments"—a hold-over from the days when bondholders turned in paper coupons to receive their interest payments.

For example, assume InDebted Corporation wants to borrow $100 million. They would issue 100,000 $1,000 bonds, setting the interest rate and maturity on them at a rate they think investors will accept (in this case, 4% for four years). Every six months, the holder of each bond will receive $20 in interest, and at the end of four years each bond holder will also receive the $1,000 principal. Every coupon payment is the same, even though the number of days between payments may vary from one period to another.

A bond holder may decide to sell his or her bond(s) before maturity. The issue becomes one of determining a fair price for the bonds. If interest rates have fallen, the bond may be worth more than the original $1,000 (the $20 payments every six months until maturity would be more attractive than the alternatives.) Conversely, if rates have risen, the bond may be worth less (since the $20 semi-annual payments are less attractive.)

One way to deal with this is to sell the bonds at what is known as a "Dutch Auction." These auctions are now held on-line. A quantity of bonds from a given issuer are offered with a stated interest rate and maturity. Bidders place bids for bonds, offering to take one or more of the bonds at either a given price (expressed as a percentage of the original $1,000 "par value"), or at a given yield (the effective annual interest rate taking into account the semi-annual interest payments and the $1,000 principal.)

The objective is to determine the "market-clearing" price—the price at which the last of the bonds would be sold. The market-clearing price is determined by ranking the offers in descending order of price, then assigning the requested number of bonds to each bidder until the last bond is sold. The price at which the last bond would be sold is the market-clearing price. Every buyer pays the market-clearing price for his or her bond(s), so everyone who is awarded bonds in the auction gets the same deal. If the total number of bonds bidders offer to buy is less than the total available for sale, every bidder will receive the bonds (s)he requested.

Continuing with our example, assume that 50 bonds of InDebted Corporation are available with a coupon rate of 4% and a maturity of three years from the effective date of the sale (known as the "settlement date.") Bonds are priced as a percentage of the $1,000 par value, with up to six digits after the decimal point. Bids might come in at the following prices:

Quantity	*Price*
20	101.96
20	101.885
15	101.78
10	101.75
5	101.6505

After sorting the list of offered prices in descending order, as has been done in the example above, bonds are assigned to bidders. The first two bidders would receive all the bonds they requested, which would take 40 of the 50 bonds available. The third bidder would receive 10 of the 15 bonds (s)he requested—this bidder also set the market clearing price at 101.78. Each bidder who was assigned bonds will pay $1,017.80 for each bond (101.78% of $1,000). The bidders lower on the list are unsuccessful and get nothing.

Problem 4
Bond Dutch Auctions (continued)

Bidders may also set their prices by offering a stated "yield to maturity" (YTM). When the bond is originally issued, the yield to maturity is the stated annual interest rate (or "coupon rate.") The yield to maturity is the equivalent rate that a bidder will receive if the bond is purchased at the offered price. If the bidder offers less than the $1,000 par value, the YTM is greater than the coupon rate; if the bidder offers more, the YTM is less than the coupon rate.

This can also be expressed as the following formula:

$$\text{Price} = \frac{c}{(1+r)} + \frac{c}{(1+r)^2} + \frac{c}{(1+r)^3} + \cdots + \frac{c}{(1+r)^n} + \frac{f}{(1+r)^n} \tag{1}$$

where c is the coupon payment in dollars, f is the par value in dollars (always 1,000), n is the number of coupon payments remaining, and r is the rate per coupon (half the YTM rate for semi-annual coupons).

Consider the four-year 4% bond described above. Table 1 shows the yields to maturity if the bidder pays $990, $1,000, and $1,010 for the bond after the bond is one year old.

Your team is to write a program that will, given the terms of a series of bond auctions and the bids for each auction, determine each auction's market clearing price and the effective yield to maturity at that price. Bonds sold at these auctions will be settled on coupon payment dates.

Input to your program will be a series of bond auctions terminated by the end-of-file. Each auction will begin with a single line containing fields separated from each other by single spaces containing the following values:

- year, month, and day of the settlement date of the auction,
- year, month, and day of the final maturity of the bond,
- the coupon annual interest rate in percent,
- and the number of bonds available.

This will be followed by a series of bids, one per line. Each bid will contain an offer to buy a quantity of bonds, the letter "p" for price or "y" for yield, and the price or yield offered. Fields will be separated from each other by single spaces. There will be at least one and at most 200 bids in any auction. The last bid will be followed by an empty line.

For each auction, your program is to print the market-clearing price (as a percentage of par value, rounded to six places after the decimal point), a single space, and that price's yield to maturity (rounded to four places after the decimal point, followed by a percent sign). No leading or trailing whitespace is to appear on an output line.

Problem 4
Bond Dutch Auctions (still continued)

Sample Input

```
2011 11 01 2014 11 01 4 50
15 p 101.780
10 p 101.750
50 p 101.650
5 p 101.480
10 p 102.005
10 y 3.883
20 p 101.885
10 p 101.960

2011 11 12 2014 11 12 4 25
10 y 4
10 p 99
10 p 101
```

Output for the Sample Input

```
101.780000 3.3712%
99.000000 4.3592%
```

Problem 4
Bond Dutch Auctions (still continued)

$1,000 4% Three-Year Bond Sold for $1,000				
Coupon	Amount	YTM	Factor	Present Value
1	20	4.00000000%	1.020000000	19.607843000
2	20		1.040400000	19.223376000
3	20		1.061208000	18.846447000
4	20		1.082432160	18.476909000
5	20		1.104080803	18.114616000
6	20		1.126162419	17.759428000
Face	1000		1.126162419	887.971382000
Total				1000.000001000

$1,000 4% Three-Year Bond Sold for $990				
Coupon	Amount	YTM	Factor	Present Value
1	20	4.35921890%	1.021796095	19.573376819
2	20		1.044067260	19.155854001
3	20		1.066823849	18.747237436
4	20		1.090076443	18.347337133
5	20		1.113835853	17.955967162
6	20		1.138113125	17.572945572
Face	1000		1.138113125	878.647278582
Total				989.999996705

$1,000 4% Three-Year Bond Sold for $1,010				
Coupon	Amount	YTM	Factor	Present Value
1	20	3.64508370%	1.018225419	19.642016028
2	20		1.036783004	19.290439680
3	20		1.055678809	18.945156263
4	20		1.074918998	18.606053142
5	20		1.094509847	18.273019704
6	20		1.114457748	17.945947288
Face	1000		1.114457748	897.297364386
Total				1009.999996491

Table 1. Table showing yields to maturity for various offer prices for the same bond.

2011/2012 SOUTHERN CALIFORNIA REGIONAL ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 5
Triangular Numbers

A *triangular number* numbers the objects that can form an equilateral triangle, as shown in the diagram in Figure 1. The nth triangle number is the number of dots in a triangle with n dots on a side; it is the sum of the n natural numbers from 1 to n. The sequence of triangular numbers is:

$$1, 3, 6, 10, 15, 21, 28, 36, 45, 55, \ldots \tag{1}$$

The triangle numbers are given by the following explicit formula:

$$T_n = \sum_{k=1}^{n} k \tag{2}$$

Your team is to write a program that will, given an integer value, determine if it is a triangular number. If the given value is a triangular number, determine the number of dots on a side.

Input to your program will be a series of lines. Each line contains an integer N, $0 < N < 10^9$ with no leading zeroes or spaces and no trailing spaces.

Your program is to determine if each integer N is a triangular number. If it is, print a line containing the number of dots on a side. If it is not a triangular number, print a line containing the string "bad". In both cases print the answer with no leading zeroes or whitespace and no trailing whitespace.

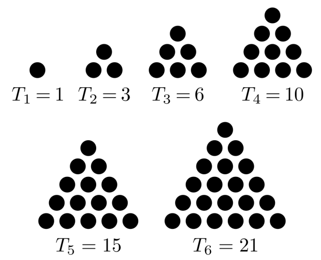


Figure 1. Diagram of the first six triangular numbers.†

Sample Input

```
55
1
587
499500
```

Output for the Sample Input

```
10
1
bad
999
```

† Text and diagram from `http://en.wikipedia.org/wiki/Triangular_number`.

2011/2012 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 6
Pavement Signage

Words and symbols painted on roads must be elongated because drivers view roads at an angle. As seen in Figure 1, a safety engineer uses an assumed eye level, h, and designates how big the symbol should appear at a distance d. The apparent size is expressed as an angle of field-of-view, α.

Your team is to write a program that will take characters and symbols represented as a series of (x, y) coordinates and produce the necessary stretched coordinates to produce a stencil for painting the image on a street surface. Referring to Figure 1, the stretched figure on the pavement (segment $\overline{\text{CE}}$) represents an unstretched image that appears as a flat image whose bottom touches the pavement at distance d (segment $\overline{\text{CD}}$).

Input to your program consists of a line describing Figure 1, followed by the vertices of one or more symbols. The first line contains three floating point values, h, d, and α, separated by whitespace. h and d are in meters, whereas α is expressed in degrees. You may assume that the road is perfectly flat, and that α will be in range such that segment $\overline{\text{AE}}$ will intersect road $\overline{\text{BCE}}$.

Symbols are (x, y) coordinate pairs, one pair per line. The x and y values are separated by whitespace. The symbol definition is terminated by a the values $x = -1$ and $y = -1$. The $(-1, -1)$ specification is *not* a part of the symbol. The last vertex in a symbol ($(0.5, 0.6)$ in the sample) connects back to the first vertex ($(0.7, 0.6)$ in the sample). There will never be more than twenty vertices in a symbol.

The technician who measures the unstretched symbol aligns it such that the leftmost point(s) touch the y-axis, and the bottom most point(s) touch the x-axis.

Output is to be a list of stretched vertices, expressed as coordinate pairs, one pair per line. Horizontal perspective is not corrected for: scale only the y values. Print the x and y values rounded to two places after the decimal point. Separate the x and y values with exactly one space. Place no leading whitespace before the x value nor any trailing white space after the y value. Print an empty line after the stretched coordinates for each symbol (including the last).

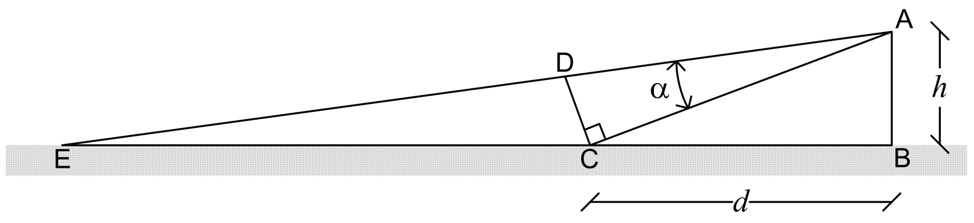


Figure 1. Diagram showing the unstretched symbol $\overline{\text{CD}}$, the road $\overline{\text{BCE}}$, h, d, and α.

Problem 6
Pavement Signage (continued)

Sample Input

```
1.5 8 6.0
 .7   .6
 .35 1
0     .6
 .2   .6
 .2  0
 .5  0
.5 .6
    -1 -1
 0 0
 0.5 0.5
 1 0
-1 -1
```

Output for the Sample Input

```
0.70 4.20
0.35 10.56
0.00 4.20
0.20 4.20
0.20 0.00
0.50 0.00
0.50 4.20

0.00 0.00
0.50 10.56
1.00 0.00
```

2011/2012 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 7
Patch Installer

Your start-up software company, though staffed with brilliant programmers, has found that errors creep into even the best production software. The company has assigned your team the task of writing an automated patch installation module to replace the current manual process.

Your team is to write a program that will, given a list of all patch identifiers (patch IDs as integers 1–999) and their relationships, match this list against one or more software configurations to produce a list of patches to apply in sequence.

Patches may have "prerequisites" (other patches that must be installed before this patch can be installed). The patch sequence must take the prerequisites into account. Since current installations have been applying patches manually, there may be systems with patch states that do not meet all the prerequisite requirements. Your program should apply all patches needed to meet any missing prerequisites. Although patch IDs are assigned in order, they are not necessarily released chronologically, so a lower patch ID may depend on patches with higher IDs. Some patch IDs may be skipped as well.

It is also possible that multiple patches may exist to address the same issue for different systems. Only one such patch can be installed in a given configuration. These appear as "conflicts" in the patch definitions. Normally, the patch with the lowest numeric patch ID should be installed, unless one of the conflicting patch IDs has already been installed manually. Conflicting patch IDs will appear as negative values in the prerequisite list for a given patch. Each conflicting patch will list the patches it conflicts with—for example, if patch 2 conflicts with patch 6, patch 6 will also conflict with patch 2.

The company's brilliant programmers consolidate patches that would be mutually dependent on each other into single patches. No two or more patches will have each other as co-requisites.

Input to your program will consist of a list of patch definitions followed by a series of software configurations. The patch definitions will consist of lines of up to 80 columns. Each line will begin with a patch ID and be followed by zero or more patch IDs of prerequisites or conflicts. Patch IDs are separated from each other by one or more spaces. The prerequisite patch IDs are listed as positive integers, and conflicting patch IDs are listed as negative integers. Every patch ID will appear in the definition list. If more than 80 columns are needed to list all the patch prerequisites and conflicts, additional lines will be used—these continuation lines will start with an ampersand in place of the numeric patch ID. This section ends with a line containing only the string "end".

The patch definitions are followed by a series of one or more software patch states. Each software patch state consists of zero or more lines containing all the patches applied so far, separated by one or more spaces and/or new-lines. A software patch state also ends with a line containing the string "end".

For each software patch state, produce a list of patches to download and apply on a single line with no leading or trailing whitespace. Print the patches in installation order, separated from each other by single spaces. If no patches need to be installed, print a line containing only the message "nothing to install".

For a given initial patch state, there may be more than one possible patch application sequence that meets the given prerequisite and conflict requirements—any sequence that meets all the requirements is acceptable.

Sample Input

```
1 2 3 4
2 4 5 -6
3
4
5
6 4 5 -2
7
8 7
9
end
1 2
 3 4
end
6
end
1 2 3 4
 5
end
7 8 9
end
```

Output for the Sample Input

```
5 7 9 8
3 4 5 7 9 8
7 9 8
3 4 5 2 1
```

2011/2012 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 8
Mall Map Maze

Swamp County Mall is the largest mall in the county. Customers keep getting lost and weary wandering around the mall looking for stores. Thankfully there is a map of the mall to help customers get around.

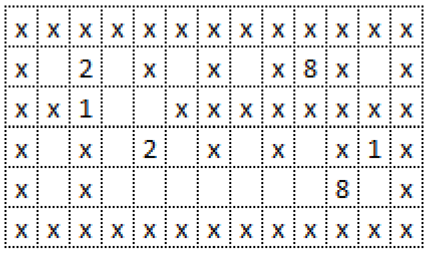
The map is in the form of a board game made up squares represented by ASCII characters. The characters in the map are as follows:

- "`x`" represents a wall or column. A square with an "`x`" cannot be entered or traversed.
- A space represents a square that can be entered.
- Digits "`1`" through "`9`" are used to designate beginning and ending points of paths.
  - Digits are located in squares that can be traversed.
  - The beginning and end of a desired path are indicated by the same digit; e. g. $(1, 1)$.
  - A given digit is in only one pair.
- There is no limit to the number of people that can be in a single available square at a given time.
- The input will start with at most a single digit in a given square.
- The board is rectangular and its edges are walls ("`x`").

Your team is to write a program that will find the shortest distance between the squares marked with the same digit. The distance is the number of moves to get from one to the other. A move is one square left, right, up, or down, but not diagonally. Thus the distance in "`xx11xx`" is one.

Input to your program will be a map in the format described above followed by end-of-file. There will be at most 100 characters in a line and at most 100 lines. Figure 1 shows the sample input as a grid for clarity.

For each digit pair in the map, your program is to print the start/end digit, one space, and the length of the shortest legal path between squares with that start/end digit. Print the length with no trailing blanks. If there is no path, report "no path" as the distance. Report the result in numerical order of the start/stop digit, lowest first. No leading or trailing whitespace is to appear on an output line.



x	x	x	x	x	x	x	x	x	x	x	x	x
x		2		x		x		x	8	x		x
x	x	1			x	x	x	x	x	x	x	x
x		x		2		x		x		x	1	x
x		x								8		x
x	x	x	x	x	x	x	x	x	x	x	x	x

Figure 1. Sample Input in grid form.

Problem 8
Mall Map Maze (continued)

Sample Input

```
xxxxxxxxxxxxx
x 2 x x x8x x
xx1  xxxxxxxx
x x 2 x x x1x
x x     8   x
xxxxxxxxxxxxx
```

Output for the Sample Input

```
1 12
2 4
8 no path
```