P1 - Prime Digital Roots

The *digital root* of a number is found by adding together the digits that make up the number. If the resulting number has more than one digit, the process is repeated until a single digit remains.

Your task in this problem is to calculate a variation on the digital root - a *prime digital root*. The addition process described above stops when there is only one digit left, but will also stop if the original number, or any of the intermediate numbers (formed by addition) are prime numbers. If the process continues and results in a single digit that is not a prime number, then the original number has no prime digital root.

> An integer *greater than* one is called a prime number if its only positive divisors (factors) are one and itself.
>
> - For example, the first six primes are 2, 3, 5, 7, 11, and 13.
> - Number 6 has four positive divisors: 6, 3, 2, and 1. Thus number 6 is *not* a prime.
> - Caveat: number 1 is *not* a prime.

EXAMPLES OF PRIME DIGITAL ROOTS

1 This is not a prime number, so 1 has no prime digital root.

3 This is a prime number, so the prime digital root of 3 is 3.

4 This not a prime number, so 4 has no prime digital root.

11 This is a prime number, so the prime digital root of 11 is 11.

642 This is not a prime number, so adding its digits gives 6 + 4 + 2 = 12. This is not a prime number, so adding again gives 1 + 2 = 3. This is a prime number, so the prime digital root of 642 is 3.

128 This is not a prime number, so adding its digits gives 1 + 2 + 8 = 11. This is a prime number, so the prime digital root of 128 is 11.

886 This is not a prime number, so adding its digits gives 8 + 8 + 6 = 22. This is not a prime number, so adding again gives 2 + 2 = 4. This is not a prime number, so 886 has no prime digital root.

INPUT FORMAT

The input will contain a single integer on each line in the range 0 to 999999 inclusive. The end of the input will be indicated by the value 0.

SAMPLE INPUT

```
1
3
4
11
642
128
886
0
```

OUTPUT FORMAT

If the input number has a prime digital root, then the input number must be output right aligned with a field width of 7. It must be followed by a single space, and then by the calculated prime digital root also right aligned with a field width of 7.

If the input number has no prime digital root, then the input number should be output as defined above followed by 4 spaces followed by the word `none` (in lowercase). The terminating zero should not be output.

SAMPLE OUTPUT

```
      1    none
      3       3
      4    none
     11      11
    642       3
    128      11
    886    none
```

P2 - Desert Bitmap

This problem requires you to search a black and white satellite image of a desert for a secret building complex with a given shape. A complex of this given shape may host an installation for producing the strategic xeenium macgillicudamate ingredient, and must keep its orientation with regard to cardinal axes (North-East-South-West). Rotations and mirror images are not allowed because they would interfere with the delicate alchemy required for the production process. You must determine how many times the given complex may possibly occur in the image.

Consider the following images, both on the same scale, where a # (sharp) is a "black" pixel representing a part of a building, and a **.** (dot) is a "white" pixel, representing sand. On the left is an image of the complex you are trying to locate, on the right is an image of the desert with some buildings on it.

```
#..
#.#
#..
```

```
#.....  #.........................#...
#.#....#.#..  #........  #............
#......#.#...##....#...##......###..
.......#.....#...#.#...##........#.#
...................#.....#.......#..
```

- How many possible locations for the given secret buildings do we count?
- The answer is *four*: one at the top-left corner, two overlapped possibilities to its right, and one in the bottom right. The shapes near the top-right corner, and in the centre bottom don't count because they are rotated (remember that rotated and/or mirrored images do not count).
- Note that, as this answer implies, the sand pixels in the image of the building complex simply establish the necessary relationships between the building parts. In the actual image they may contain *either* sand *or* other building parts (possibly for disguising the true nature of the complex).
- *Assume* that images representing strategic complexes are already trimmed of any unneeded dot "white" pixels on the edges, ie, these images will always contain *at least one # character on each edge* (as our example shows). An edge here is the first or last row or column.

INPUT FORMAT

Each problem will give you the specification for the building complex image followed by the specification for the desert image. There may be several problems in the input data, which will be terminated by a line containing just 0 0.

In each problem the input is:

- Line 1: 2 positive integers, *B1*, *B2*, respectively representing the number of lines and the numbers of columns in the following *B*uildings image. Both numbers will be in the range 1 to16 inclusive.
- Next *B1* lines: *B2* characters (# or **.**) on each line to represent part of the image of the building complex.
- Next Line: 2 positive integers, *D1*, *D2*, respectively representing the number of lines and the numbers of columns in the following *D*esert image. Both numbers will be in the range 1 to 64 inclusive.
- Next *D1* lines: *D2* characters (# or **.**) on each line to represent the desert image.

SAMPLE INPUT

```
2 2
#.
##
3 5
#.#.#
#####
.###.
1 3
#.#
3 6
##..##
.#.#.#
#.#...
3 3
#..
#.#
#..
5 36
#......#.......................#...
#.#....#.#...#.........#...........
#......#.#...##....#...##......###..
.......#.....#...#.#...##........#.#
...................#.....#.......#..
0 0
```

OUTPUT FORMAT

The output will consist of one integer for each problem given, on a line by itself, with no spaces, giving the number of possible occurrences of the given building complex.

SAMPLE OUTPUT

```
4
3
4
```

P3 - Normalized Histogram

A histogram is a bar graph, and a convenient way to view a distribution of values. Each bar of the graph represents a range of values, and the length of a bar represents the number of input values in that range. Your program must read a collection of non-negative integer input values, and display it as a histogram. The bars of the histogram are represented as horizontal rows of "#" characters. The parameters of the histogram are:

- the *lo* bound (the lowest value that the histogram can represent)
- the *step* size (the range of values represented by a single bar)

The bars of the histogram are numbered from 0 up to some maximum number. The number of "#" characters in histogram bar number *n* is the number of input values which lie between *lo+n*step* and *lo+(n+1)*step-1*. So bar 0 of a histogram where *lo* is 30 and *step* is 10 will represent values between 30 and 39, bar 1 of the same histogram will represent values from 40 to 49.

For this problem, all input values, and the histogram parameters, are non-negative integers. Histogram parameters must be chosen to be "nice" round numbers that will allow the histogram to have no more than 20 bars. The constraints are as follows:

- The *step* size must be 1 or 2 or 5, possibly multiplied by some positive integer power of 10. For example: 2, 100, 5000. The *step* size is thus an integer.
- The *lo* bound must be some multiple of the *step* size.
- The *lo* bound and the *step* size must be chosen so that the minimum input value counts towards the first histogram bar, and the maximum input value counts towards a histogram bar number which is less than 20, but otherwise is as large as possible. That is, the histogram *has as many bars as possible*, up to a *maximum* of 20.

The input to your program consists of a series of data sets. Each data set begins with a single integer in the range 2 to 500 inclusive, on a line by itself. This is the number of input values. It is followed by the non-negative integer data values, which may be split across lines arbitrarily. The input values may or may not be sorted. The entire input is terminated by an empty data set (where the initial count is zero).

For each data set, the program must print out the corresponding histogram. The histogram is a series of histogram bars, one to a line. Each bar is preceded by the bottom of the range of values for that bar, displayed right justified in a field eight characters wide. This bound is followed by one space, and then a row of "#" characters, corresponding to the number of input values for this histogram bar. The input data is such that no bar will have more than 70 "#"'s. The histogram should be printed from the first non-empty bar to the last non-empty bar, inclusive, and should be followed by one blank line.

As a special case, if all input values in a given set are equal, then no histogram is displayed, and the program must give a line exactly in the following form:

```
All inputs equal 15.
```

There must be exactly three spaces at the positions indicated, and a full stop at the end of the line. As with a valid histogram, this message must be followed by a blank line.

SAMPLE INPUT

```
55
  0  10  20  30  40  50  60  70  80  90
  0  10  20  30  40  50  60  70  80  90
 81  82  83  84  85  86  87  88  89  81
 81  82  83  84  85  86  87  88  89  81
150 200 250 300 350 400 450 500 550 600
700 750 800 850 900
12
11 11 11 11 11 11 11 11 11 11 11 11
2
5001 3099
0
```

SAMPLE OUPUT

```
       0 ##########
      50 ##############################
     100
     150 #
     200 #
     250 #
     300 #
     350 #
     400 #
     450 #
     500 #
     550 #
     600 #
     650
     700 #
     750 #
     800 #
     850 #
     900 #

All inputs equal 11.

    3000 #
    3200
    3400
    3600
    3800
    4000
    4200
    4400
    4600
    4800
    5000 #
```

P4 - Encryption Scheme

Most text encryption schemes use a secret key string to convert the plain text to the enciphered text in some way. A novel method being tested by the Australian Security Service consists of a transformation of a key string *K* into a target string *P* using block moves. Each *block move* is of the form *copy*(*start*, *length*), where *start* indicates a position in *K* and *length* is the number of characters to be copied from *K* to *P*. Since the idea is to eventually transmit only the block moves, the principle is to use as few block moves as possible. For example if:

```
K: abaabba
P: aaabbbabbbaaa
```

Assuming that here string positions start with 1, two shortest block move sequences would be:

copy(3,2);*copy*(4,3);*copy*(2,2);*copy*(5,2);*copy*(2,3);*copy*(1,1)

or

copy(7,1);*copy*(3,3);*copy*(5,2);*copy*(4,2);*copy*(5,3);*copy*(3,2)

The actual shortest block move sequences are not unique but the minimum number is, 6 in this case. If the moves are now transmitted, then it is possible to construct the plaintext message *P* from the key string *K*.

The Australian Security Service is now automating this procedure, so given *K* and *P* they need to count the minimum number of block moves from *K* to *P*. To make things simple at the beginning, they are considering strings comprised of lowercase letters and digits. The set of characters within string *P* is a subset of the set of characters of the key string *K*.

You are to help the Australian Security Service by writing a program to get two strings *K* and *P* as above, and print the minimum number of block moves from *K* to *P*. Your code will be tested with a sequence of lines. Odd lines are to be used as the key strings *K*, and even lines to be used as target strings *P*. The output will consist solely of the minimum number of block moves for each pair. The input will be terminated by a '#' by itself in the place of a *K* string.

Assume that each of *K* and *P* is made up of 1 to 120 characters (*K* is allowed to be longer than *P*).

SAMPLE INPUT

```
abaabba
aaabbbabbbaaa
xy0z
zzz0yyy0xxx
#
```

SAMPLE OUTPUT

```
6
10
```

COMMENTS

The first sample is discussed on the first page. Here follows a minimal sequence of block moves for the second sample:

copy(4,1);*copy*(4,1);*copy*(4,1);*copy*(3,1);
copy(2,1);*copy*(2,1);*copy*(2,2);
copy(1,1);*copy*(1,1);*copy*(1,1)

P5 - James Bond

A terrorist hides in an underground sewage system. Any two pipes have at most one common node that is also an endpoint for both of them. The terrorist hides in one such node and has placed clock-activated bombs at several other nodes. The passage into and through a node becomes impossible after the explosion.

James Bond wants to capture this terrorist and has at his disposal a complete map of the sewage system that also highlights the position of the terrorist, the placement of the bombs, and their timings. At time 0 Bond starts from one of the nodes and must reach the terrorist in the shortest possible time. All pipes are bi-directional, at least for Bond. He can travel both ways with the same speed.

Bond dies if an explosion catches him in one of the trapped node, but otherwise he is unscathed and can pursue his search (even if he is just one second away from the blast).

Determine the minimum time that Bonds needs to capture the terrorist, if this is possible.

INPUT FORMAT

- Line 1: Four positive integers N, M, S, T. N is the number of nodes and nodes are numbered 1, 2, … N. M is the number of pipes. There is at most one pipe between each pair of nodes. S is Bond's starting node, and T is the node where the terrorist hides ($1 \le S, T \le N \le 100$, $1 \le M \le N^2$).

- Next N Lines: Each line contains 0 if the corresponding node is not trapped. Otherwise it contains a positive integer X giving the time when the bomb will detonate ($1 \le X \le 1000$).

- Next M Lines: Each line contains two node numbers representing the pipe's end nodes, and a positive integer giving the travel time Y between its end nodes ($1 \le Y \le 1000$).

The above set can be repeated several times. The end is signaled by a line with $N = M = S = T = 0$.

OUTPUT FORMAT

Print one line for each input set, giving the minimum time that Bonds needs to capture the terrorist, if this is possible (thus print 0 if Bond starts at the terrorist node). Otherwise print 0.

SAMPLE INPUT

```
4 4 1 4
0
0
5
0
2 1 3
3 4 4
3 2 2
1 3 4
3 2 1 3
0
1
0
1 2 3
2 3 1
0 0 0 0
```

SAMPLE OUTPUT

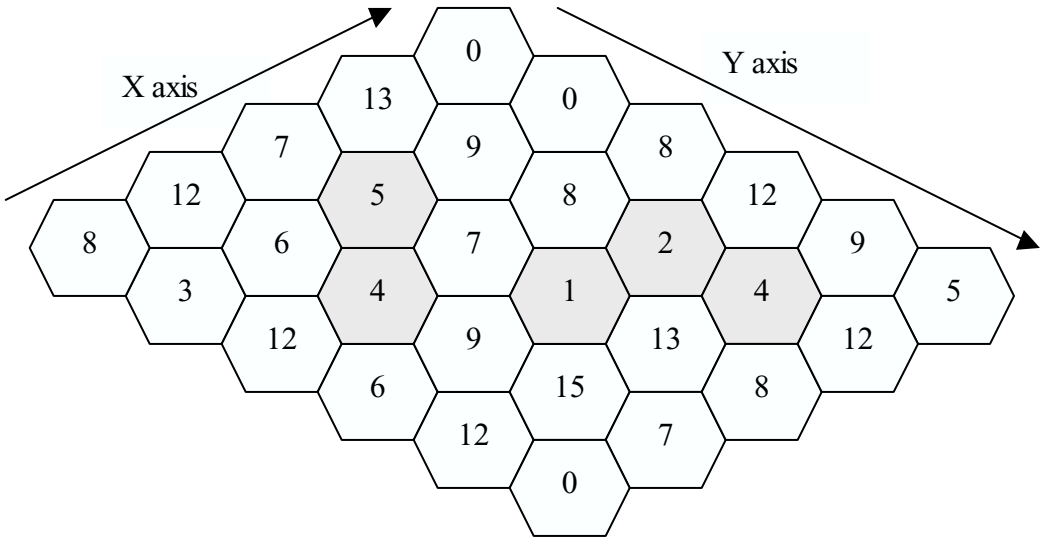
```
8
0
```

P6 - Basalt Buckets

Under certain conditions volcanic basalt forms large crystals, like hexagonal pillars. At Fingal Head, at the border of New South Wales and Queensland, there is a dramatic example, called Giant's Causeway, where a peninsula formed of such columns juts into the Pacific. It is particularly dramatic when the big Pacific rollers break on the causeway, leaving streams of water cascading over the basalt pillars.

Your task is to find out how much water could collect in hexagonal hollows, formed when some pillars are shorter than others, and can act as wells.

Here is a diagram of a set of hexagons. Each hexagon has an integer height. Water can always cascade off the edge of the set of hexagons, but it will collect in the five shaded hexagons, since they form wells completely surrounded by higher hexagons. Water drains from the left hand pair over two pillars of height six, and from the three on the right water drains over a pillar of height seven. Assuming each hexagon has unit area, the volume of water that can collect is 17 units.



Your program must handle input as a series of problem descriptions.

Each problem begins with two integers *X* and *Y*, on a line by themselves, in the range 1 to 200 inclusive, giving the number of hexagons along an *X*-axis, and along a *Y*-axis, as shown in the diagram. Then there are given a further *X***Y* integers, in the range 0 to 5000 inclusive, which are the heights of the hexagons. The order of input heights is given as *Y* rows of *X* integers, but they may be split across lines arbitrarily. There may be several problems in the input data, which will be terminated by a line containing just 0 0.

The given diagram is described by input as follows.

SAMPLE INPUT

```
5 6
8 12 7 13 0 3 6 5 9 0 12
4 7 8 8 6 9 1 2 12 12 15 13 4 9 0 7 8 12 5
0 0
```

The output must consist only of one integer for each problem given, on a line by itself with no spaces, giving the volume of water that could collect for that set of columns. The output for the given problem description would thus be:

SAMPLE OUTPUT

```
17
```

If in the given diagram the shades are not clearly visible, here follow the *X* and *Y* coordinates of the shaded hexagons, with their respective heights:

X	Y	Height
3	2	5
2	3	4
3	4	1
4	4	2
4	5	4

P7 - Take 5

In certain entertainment magazines there is a type of crossword puzzle where, instead of clues for words, the cells contain positive integers. There is a hidden one-to-one correspondence with numbers and letters, and the goal of the puzzle is to assign alphabet letters to these positive numbers so that sequences of two or more letters (across the rows and down the columns) form valid words.

Your task is to write a program to help solve this type of puzzles.

The input will consist of a series of games. Each game will have a valid dictionary of words (preceded by a size) that can be used, followed by a puzzle grid (preceded by *r* and *c*, the number of rows and columns). Each word will be at least 2 and at most 10 characters in length from the set 'a' – 'z'. The dictionary size will be at most 1000 words; the dictionary entries are unique but not necessarily sorted. Each puzzle grid consists of *c* columns, $1 <= c <= 10$, and *r* rows, $1 <= r <= 10$, of cell numbers in the range 0-26. A positive cell number denotes a letter and a cell number of 0 denotes a ' ' (blank character). The numbers form a continuous range, thus one can expect numbers such as 0, 1, 2, 3, 4, 5 but not 0, 1, 12, 13, 24, 25.

The series of games is terminated by a "game" with dictionary size 0 and should not be processed.

The output should consist of the *guaranteed uniquely* solved puzzle for each game, where the cell numbers are replaced with characters 'a' – 'z', ' '. A blank line should separate output solutions.

SAMPLE INPUT

```
3
max
min
nix
3 4
1 2 3 0
4 0 0 0
5 4 3 0
12
boon
boonbar
boony
foobar
foorag
goon
goony
rannoo
ranoon
toon
toonbar
toony
6 6
1 2 2 3 4 5
2 0 0 2 0 0
2 0 0 2 0 6
5 4 7 7 2 2
4 0 0 8 0 2
9 0 0 0 0 7
0
```

SAMPLE OUTPUT

```
max
i
nix

foobar
o  o
o  o t
rannoo
a  y o
g    n
```

P8 - Maximum Subsequence

Background definitions

Subsequence of a given string: Any string that can be obtained by deleting zero or more symbols from the given string (the remaining symbols occur in the same order, but aren't necessarily consecutive).

Examples: "", "a", "xb", "aaa", "bbb", "xabb", "xaaabbb" are subsequences of "xaaabbb".

Common subsequence of a given set of strings: Any string that is a subsequence of each of the given strings.

Examples: "xa", "aaa", "bbb" are common subsequences of the strings in {"xaaabbb", "a7axb8bab", "bbabartxta"}.

Longest common subsequence of a given set of strings: Any *maximum length* common subsequence of the given set of strings.

Example: "aaa", "bbb" are the two longest common subsequences of the above set of strings. Their length is 3, which is greater than the length of any other common subsequence.

Determine the maximum length of the common subsequences of a given set of N non-empty words.

Consider a set of N words (strings), where each word is a non-empty sequence of lowercase English letters and digits. Assume that $2 \le N \le 100$, that each word contains maximum 100 characters, and that in each set the product of all its word lengths is maximum 2,097,152.

INPUT FORMAT:
The input may consist of several sets, and each set consists of several lines.

- Line 1: A positive integer N giving the number of words in the current set. The value $N = 0$ signals the end.

- Next N Lines: Each line consists of a non-empty word, ie, a non-empty sequence of lowercase English letters and digits.

OUTPUT FORMAT:
The output consists of one line for each input set.

- Each line contains an integer giving the maximum length of all common subsequences of the corresponding set (in input order).

SAMPLE INPUT:

```
2
ab
bc
3
xaaabbb
a7axb8bab
bbabartxta
3
ab
bc
cd
2
1abc2def3ghi4jkl5mno6pqr7stu8vwx9yz0
abc8def7ghi6jkl0mno4pqr3stu2vwx1yz
0
```

SAMPLE OUTPUT and COMMENTS

```
1
3
0
26
```

```
1 = the length of “b”
3 = the length of either “aaa” or “bbb”
0 = the length of the empty word
26 = the length of “abcdefghijklmnopqrstuvwxyz”
```

P9 - Kth Smallest

Consider arithmetic-type formulas consisting of the following symbols:

- uppercase letters in the 'A'-'L' range;
- two operators '+', '-'; and
- round parentheses '(', ')'.

The letters represent variables with integer values. Our two operators are a bit non-standard here: the '+' operator denotes the *maximum*, and the '-' operator denotes the *minimum* of the two operands. Both operators are left-associative, and the *minimum* operator has a higher priority than the *maximum* operator. Parentheses are optional and are used for grouping in the usual way. Formula representations may also contain white spaces, which are ignored.

For example, the following two formulas are equivalent in our interpretation:

```
A-B + A-C + B-C
```

and

```
A-B + (A-C)    +    ((C  -  (   (B)   )))
```

According to our rules, both these formulas denote the same expression:

max(*max*(*min*(A,B), *min*(A,C)), *min*(C, B))

The above expression will always return the 2nd smallest value in the *multiset* {*A*, *B*, *C*}, regardless of the numerical values assigned to its variables. For example, consider $A = 50$, $B = 70$, $C = 30$. The value returned by the above expression is 50, which is the 2nd smallest value in {50, 70, 30}. Consider $A = 50$, $B = 30$, $C = 30$. The value returned by this expression is 30, which is the 2nd smallest value in the multiset {50, 30, 30}.

> A multiset is like a set except that it can have repetitions of identical elements.

Determine if a given formula always returns the *K*th smallest value regardless of the values assigned to its variables, for a given *K*. Our formulas are *guaranteed* valid. The formulas contain between 1 and 12 different variables (but not necessarily consecutive), and at most 20,000 total symbol occurrences. *K* will be in the range 1 to the number of different variables actually used.

INPUT FORMAT

The input contains a sequence of formulas. Each formula starts with a positive integer *K* at the beginning of a line, followed by a space, and then a valid formula containing the following characters: 'A'-'L', '+', '-', '(', ')', and ' ' (space). Each line contains a maximum of 120 characters. Long formulas are broken over several lines, and the continuation is signaled by an '_' (underscore) at the end of every line except the last. The value $K = 0$ signals the end of the input.

OUTPUT FORMAT:

The output contains one line for each input formula. Each line consists of either the word YES if the formula satisfies our request, or NO otherwise (both in uppercase).

SAMPLE INPUT

```
1 A
1 (A)
1 ((A))
1 A+B
1 A-B
2 A+B+C
2 A-B-C
2 A-B + A-C + B-C
2 A-B + (A-C)    +_
    ((C  -  (   (B)   )))
2 (A+B) - (A+C) - (B+C)
2 (A+I) - (A+L) - (I+L)
0
```

SAMPLE OUTPUT

```
YES
YES
YES
NO
YES
NO
NO
YES
YES
YES
YES
```