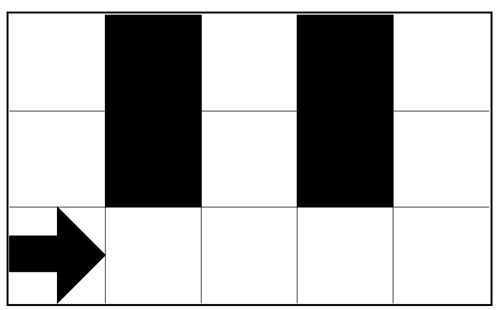
# Problem S: Amazing

One of the apparently intelligent tricks that enthusiastic psychologists persuade mice to perform is solving a maze. There is still some controversy as to the exact strategies employed by the mice when engaged in such a task, but it has been claimed that the animal keepers eavesdropping on conversations between the mice have heard them say things like "I have finally got Dr Schmidt trained. Every time I get through the maze he gives me food".

Thus when autonomous robots were first being built, it was decided that solving such mazes would be a good test of the 'intelligence' built into such machines by their designers. However, to their chagrin, the first contest was won by a robot that placed a sensor on the right hand wall of the maze and sped through the maze maintaining contact with the right hand wall at all times. This led to a change in the design of the mazes, and also to interest in the behaviour of such robots. To test this behaviour the mazes were modified to become closed boxes with internal walls. The robot was placed in the south west corner and set off pointing east. The robot then moved through the maze keeping a wall on its right at all times. If it cannot proceed it will turn left until it can proceed. All turns are exact right angles. The robot stops when it returns to the starting square. The mazes were always set up so that the robot could move to at least one other square before returning. The researchers then determined how many squares were not visited, and how many were visited once, twice, thrice and four times. A square is visited if a robot moves into and out of it. Thus for the following maze, the values (in order) are: 2, 3, 5, 1, 0.



Write a program to simulate the behaviour of such a robot and collect the desired values.

Input will consist of a series of maze descriptions. Each maze description will start with a line containing two integers representing the size of the maze (b and w; $1 \leq b \leq 10$, $1 \leq w \leq 10$). This will be followed by b lines, each consisting of w characters, either "0" or "1". Ones represent closed squares, zeroes represent open squares. Since the maze is enclosed, the outer wall is not specified. The file will be terminated by a line containing two zeroes.

Output will consist of a series of lines, one for each maze. Each line will consist of 5 integers representing the desired values, each value right justified in a field of width 3.

## Sample input

```
3 5
01010
01010
00000
0 0
```

## Sample output

```
  2  3  5  1  0
```

## Problem T: Minary Encoding

The speed of the standard binary multiplication algorithm depends on the number of non-zero bits in the multiplier. If we introduce a 'negative' bit, then strings of 2 or more '1' bits can be replaced by a '1' bit, a string of '0' bits and a 'negative' bit. If we represent the 'negative' bit by '#', then the string '01111' can be replaced by '1000#'. This string has only two non-zero bits as opposed to 4 in the original formulation. We call the resultant string a 'minary' string.

Write a program that will read in a series of strings, in either binary or minary notation and will convert them to the other notation. Note that on input, all binary strings will start with '0', and all minary strings will start with '1'. This will not necessarily be true of the output.

Input will consist of a series of lines, each line consisting of either a binary number (starting with '0') or a minary number (starting with '1'). 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 line will consist of a converted string, that is binary numbers are represented in minary, and vice versa.

### Sample input

```
01110
0111101111
100#010100#
#
```

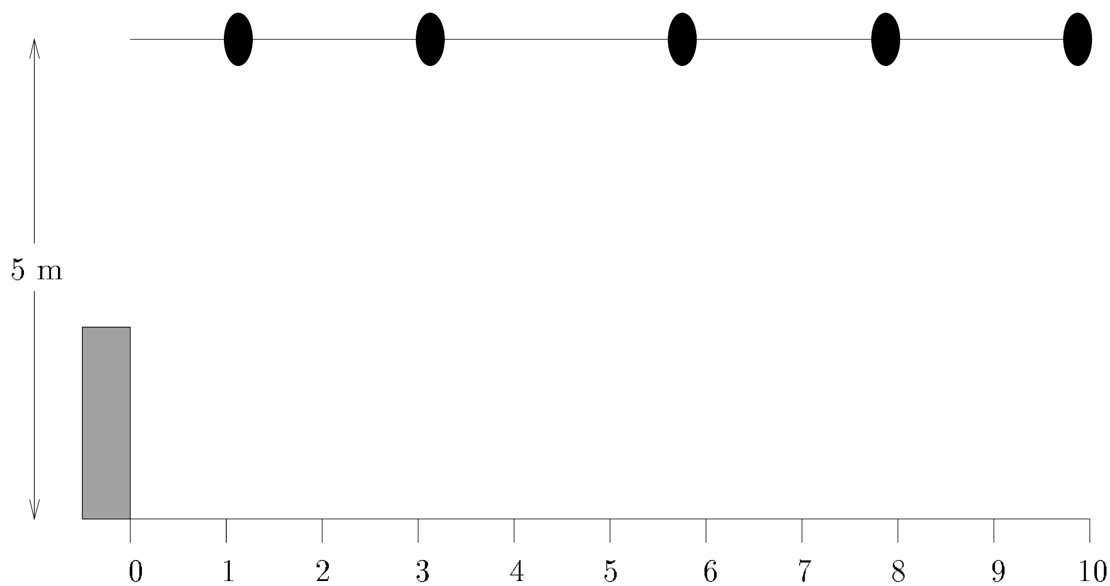
### Sample output

```
100#0
1000#1000#
01110100111
```

## Problem U: Keeping Dry

Most of us have faced the problem of walking home in the rain. The problem is simply that we don't know when we are going to be hit by a falling raindrop—if we knew when the drops were coming, and were agile enough (and thin enough), we could dodge the rain and avoid getting wet even in heavy showers. The Engineering Department at Waikikamukau Polytech believe they have found the solution, however. They have developed a radar detector to spot raindrops at a height of 5 metres. This information is fed into a computer, which calculates a strategy for avoiding the drops. To test their strategy, they have developed a one-dimensional model, for an experimental track of length 10 metres. The person is simulated by a rectangle of height 2 metres and width 0.5 metres.

Write a program to test their experimental drop-dodging strategy. Your program will read in successive positions of the 'person', together with the positions (x-coordinates) of all the raindrops at a height of 5 metres above the ground at that time, and determine how many raindrops ultimately hit the person. Raindrops fall at a constant rate of 1 metre per second.




Input will consist of a series of scenarios, each scenario consisting of a series of lines. Each line will give the data at one second intervals. The first value on a line gives the position of the front of the 'person' along the track, the remaining values give the positions of the new raindrops which have been detected at the 5m height. Two drops will never have the same height and position. All positions are given to the nearest cm (0.01 m). Each line is terminated by the number $-1.00$. You can assume all drops are at the 5m height at the start of the second when they are first detected, and fall at 1m per second. Each scenario will be terminated by a line containing the single value 10.00, indicating that the 'person' has reached the end of the track. The entire file will be terminated by $-1.00$.

Output will consist of a series of lines, one for each scenario, each containing the number of raindrops that hit the 'person'.

### Sample input

```
0.00 1.05 3.06 5.81 7.93 9.91 -1.00
2.10 1.87 4.21 6.83 8.76 -1.00
1.63 2.44 6.17 8.13 9.45 -1.00
8.25 2.83 3.61 4.77 5.56 7.31 8.11 9.23 9.84 -1.00
10.00
-1.00
```

### Sample output

```
2
```

## Problem V: Power Crisis

During the power crisis in New Zealand this winter (caused by a shortage of rain and hence low levels in the hydro dams), a contingency scheme was developed to turn off the power to areas of the country in a systematic, totally fair, manner. The country was divided up into N regions (Auckland was region number 1, and Wellington number 13). A number, m, would be picked 'at random', and the power would first be turned off in region 1 (clearly the fairest starting point) and then in every m'th region after that, wrapping around to 1 after N, and ignoring regions already turned off. For example, if N = 17 and m = 5, power would be turned off to the regions in the order:1,6,11,16,5,12,2,9,17,10,4,15,14,3,8,13,7.

The problem is that it is clearly fairest to turn off Wellington last (after all, that is where the Electricity headquarters are), so for a given N, the 'random' number m needs to be carefully chosen so that region 13 is the last region selected.

Write a program that will read in the number of regions and then determine the smallest number m that will ensure that Wellington (region 13) can function while the rest of the country is blacked out.

Input will consist of a series of lines, each line containing the number of regions (N) with $13 \leq N < 100$. The file will be terminated by a line consisting of a single 0.

Output will consist of a series of lines, one for each line of the input. Each line will consist of the number m according to the above scheme.

### Sample input

```
17
0
```

### Sample output

```
7
```

## Problem W: Recycling

Kerbside recycling has come to New Zealand, and every city from Auckland to Invercargill has leapt on to the band wagon. The bins come in 5 different colours—red, orange, yellow, green and blue—and 5 wastes have been identified for recycling—Plastic, Glass, Aluminium, Steel, and Newspaper. Obviously there has been no coordination between cities, so each city has allocated wastes to bins in an arbitrary fashion. Now that the government has solved the minor problems of today (such as reorganising Health, Welfare and Education), they are looking around for further challenges. The Minister for Environmental Doodads wishes to introduce the "Regularisation of Allocation of Solid Waste to Bin Colour Bill" to Parliament, but in order to do so needs to determine an allocation of his own. Being a firm believer in democracy (well some of the time anyway), he surveys all the cities that are using this recycling method. From these data he wishes to determine the city whose allocation scheme (if imposed on the rest of the country) would cause the least impact, that is would cause the smallest number of changes in the allocations of the other cities. Note that the sizes of the cities is not an issue, after all this is a democracy with the slogan "One City, One Vote".

Write a program that will read in a series of allocations of wastes to bins and determine which city's allocation scheme should be chosen. Note that there will always be a clear winner.

Input will consist of a series of blocks. Each block will consist of a series of lines and each line will contain a series of allocations in the form shown in the example. There may be up to 100 cities in a block. Each block will be terminated by a line starting with 'e'. The entire file will be terminated by a line consisting of a single #.

Output will consist of a series of lines, one for each block in the input. Each line will consist of the number of the city that should be adopted as a national example.

### Sample input

```
r/P,o/G,y/S,g/A,b/N
r/G,o/P,y/S,g/A,b/N
r/P,y/S,o/G,g/N,b/A
r/P,o/S,y/A,g/G,b/N
e
r/G,o/P,y/S,g/A,b/N
r/P,y/S,o/G,g/N,b/A
r/P,o/S,y/A,g/G,b/N
r/P,o/G,y/S,g/A,b/N
ecclesiastical
#
```

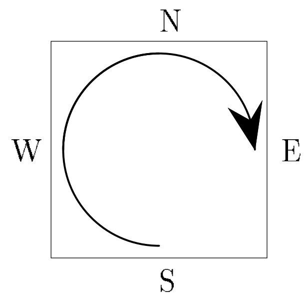
### Sample output

```
1
4
```

# Problem X: Bridge Hands

Many games, such as Bridge, involve dealing a standard deck of 52 cards to 4 players, so that each receives 13 cards. Good players can then play with the hand as it is dealt, but most ordinary players will need to sort it, firstly by suit, and then by rank within suit. There is no fixed ranking of the suits for this purpose, but it is useful to alternate the colours, so we will presume the following ordering: $\clubsuit < \diamondsuit < \spadesuit < \heartsuit$. (Note that because most character sets do not recognise these symbols, from now on we will use the more conventional C, D, S, H). Within a suit, Ace is high, so the ordering is $2 < 3 < 4 < 5 < 6 < 7 < 8 < 9 < T < J < Q < K < A$.

The players are usually designated North, South, East and West, and they sit at the points of the compass they name. One player is designated the dealer and she deals one card to each player starting with the player on her left and proceeding clockwise until she deals the last card to herself.




Write a program that will read in a representation of a deck of cards, deal them, sort them, and then display the 4 sorted hands in the format shown below.

Input will consist of a series of deals. Each deal will consist of the letter representing the dealer (N, E, S, W) followed by two lines representing the deck as shown below. The file will be terminated by a line consisting of a single #.

Output will consist of a series of sets of four lines, one set for each deal. Each set will consist of four lines displaying the sorted hands, in the order and format shown below. Sets must follow each other immediately, with no blank line between them.

## Sample input

```
N
CQDTC4D8S7HTDAH7D2S3D6C6S6D9S4SAD7H2CKH5D3CTS8C9H3C3
DQS9SQDJH8HAS2SKD4H4S5C7SJC8DKC5C2CAHQCJSTH6HKH9D5HJ
#
```

## Sample output

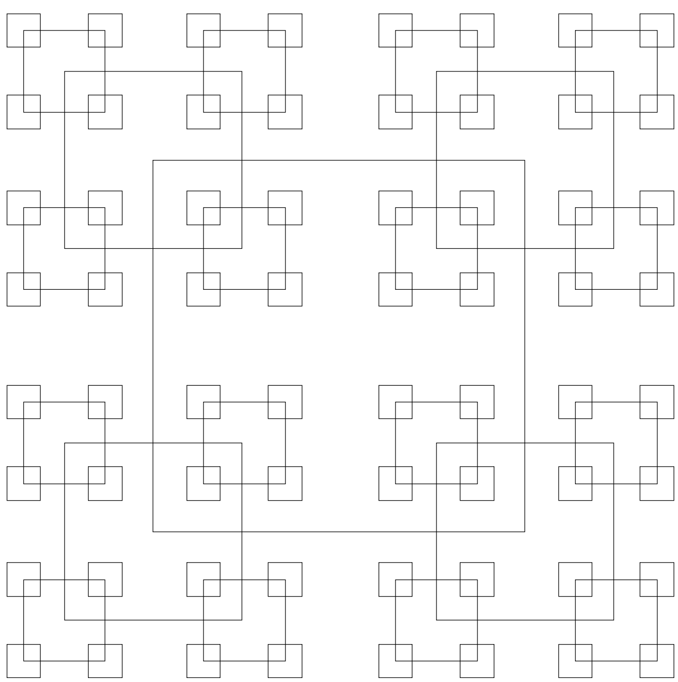
```
S: C3 C5 C7 CT CJ D9 DT DJ S3 SK H2 H9 HT
W: C2 C4 CK D4 D5 D6 DQ DA S4 S8 ST SJ H8
N: C6 C8 C9 CA D8 S9 SA H4 H5 H6 H7 HJ HA
E: CQ D2 D3 D7 DK S2 S5 S6 S7 SQ H3 HQ HK
```

## Problem Y: All Squares

Geometrically, any square has a unique, well-defined centre point. On a grid this is only true if the sides of the square are an odd number of points long. Since any odd number can be written in the form 2k+1, we can characterise any such square by specifying k, that is we can say that a square whose sides are of length 2k+1 has size k. Now define a pattern of squares as follows.

1. The largest square is of size k (that is sides are of length 2k+1) and is centred in a grid of size 1024 (that is the grid sides are of length 2049).
2. The smallest permissible square is of size 1 and the largest is of size 512, thus $1 \leq k \leq 512$.
3. All squares of size $k > 1$ have a square of size k div 2 centred on each of their 4 corners. (Div implies integer division, thus 9 div 2 = 4).
4. The top left corner of the screen has coordinates (0,0), the bottom right has coordinates (2048, 2048).

Hence, given a value of k, we can draw a unique pattern of squares according to the above rules. Furthermore any point on the screen will be surrounded by zero or more squares. (If the point is on the border of a square, it is considered to be surrounded by that square). Thus if the size of the largest square is given as 15, then the following pattern would be produced.



Write a program that will read in a value of k and the coordinates of a point, and will determine how many squares surround the point.

Input will consist of a series of lines. Each line will consist of a value of k and the coordinates of a point. The file will be terminated by a line consisting of three zeroes (0 0 0).

Output will consist of a series of lines, one for each line of the input. Each line will consist of the number of squares containing the specified point, right justified in a field of width 3.

## Sample input

```
500 113 941
0 0 0
```

## Sample output

```
  5
```

## Problem Z: Ticket Pricing

With the current hard times, many impresarios are rethinking their ticket pricing policies. One startegy is as follows: "Currently I am selling 100 tickets at $30.00 each. If I dropped the price by 50c I would sell 15 more. However, our costs are currently $10.00 per head, and these would increase by 20c per head. Thus at the moment I sell 100 tickets at $30.00 giving $3000, less 100 times $10.00 giving a profit of $2000. If I cut the price I would sell 115 tickets at $29.50 giving $3392.50 less 115 times $10.20 giving a profit of $2219.50. Wow, I wonder how much further I can go?". If he extends this example, he could produce a table similar to the following:

| Price($) | Ticket Sales | Overheads($) | Profit($) |
|---|---|---|---|
| 30.00 | 100 | 10.00 | 2000.00 |
| 29.50 | 115 | 10.20 | 2219.50 |
| 29.00 | 130 | 10.40 | 2418.00 |
| ... | ... | ... | ... |
| 24.50 | 265 | 12.20 | 3259.50* |
| 24.00 | 280 | 12.40 | 3248.00 |
| ... | ... | ... | ... |

This is an example of a break-even chart. It is characterised by 6 numbers: the current selling price, number of tickets sold and overheads together with the projected changes in these quantities. The line marked * gives rise to the maximum profit, and the price on that line is called the optimum selling price.

Write a program that will read in a series of ticket pricing situations and determine for each the optimum selling price and the expected profit at that price. If there is more than one price that produces a maximum profit, choose the lowest—you never know, you may sell more tickets that way. Prices must always be greater than zero (you are not a charity). The mood of the public is such that price rises are not even contemplated—if the current selling price is better than anything lower, well so be it. It is possible that different price reductions may produce a better overall profit (in the above case a price drop of about 47.5 cents is better), but your brain (and that of your accountant) is not able to handle such subtleties.

Input will consist of a series of lines, each line consisting of 6 values as outlined above. Note that all numbers will be strictly positive (> 0) although the difference in price must be taken as a **reduction**. The file will be terminated by a line consisting of 6 zeroes (0 0 0 0 0 0).

Output will consist of a series of lines, one for each line of the input. Each line will consist of the optimum ticket price and the expected profit at that price. There must be one (1) space between the two values.

### Sample input

```
30.00 100 10.00 0.50 15 0.20
0 0 0 0 0 0
```

### Sample output

```
24.50 3259.50
```