1999-2000 ACM North Central Regional Programming Contest
Problem Statements

The problem statements for the 1999-2000 ACM North Central Regional Programming Contest can be viewed by clicking on the links given below. The HTML representation of the problems was created directly from the Microsoft Word documents used for the versions teams and judges used during the contest. The only exception is that minor changes were made to reflect the effect of the clarifications issued during the contest.

1. Gray Codes
2. Cycles, Period.
3. Wild Cards
4. Clock Angles
5. Going Down!
6. Hamming It Up

1999-2000 ACM North Central Regional Programming Contest

Problem 1 — Gray Codes

One description of a Gray code (named after xxx) is that it is a sequence of 2^k binary numbers, each having k digits, such that adjacent numbers in the sequence differ in only one digit (or bit position). For example, consider the case $k = 3$. A sequence of eight 3-bit numbers satisfying the requirements is shown in *Figure 1*, below. Note that the sequence is cyclic, in that the condition that only a single bit is different between adjacent numbers in the sequence is true when considering the last (**1 0 0**) and the first (**0 0 0**) numbers in the sequence.

0	**0**	**0**						**0**	**0**	**0**
0	**0**	**1**						**0**	**0**	**1**
0	**1**	**1**						**0**	**1**	**0**
0	**1**	**0**						**0**	**1**	**1**
1	**1**	**0**						**1**	**0**	**0**
1	**1**	**1**						**1**	**0**	**1**
1	**0**	**1**						**1**	**1**	**0**
1	**0**	**0**						**1**	**1**	**1**
Figure 1				*Figure 2*				*Figure 3*		

A classic application of Gray codes is in sensing the position of a drum attached to a rotating mechanical object. Imagine that a strip of paper having k bands (three in this example) is prepared and attached to the drum. Each band is subdivided into 2^k colored cells, with a white cell representing a 0 bit in the code, and a black cell representing 1. The three bands for the case $k = 3$ would appear as shown in *Figure 2*. Associated with each strip is a light sensor that reports 0 when it senses white, and 1 when it senses black. If the strip is attached to the rotating object, then the sensors report the code (as a 3-bit number) corresponding to the approximate rotational position of the object. Since only one sensor's output changes at a time during rotation, there is no possibility that imprecise sensor placement will result in an incorrect rotational position report.

Examining *Figure 1* we can better understand this physical interpretation of Gray codes. Since there are 8 codes corresponding to 360 degrees of rotation, we see that each code corresponds to a 45-degree region. For example, when the sensors report **000** we know the device is in the 0 to 45 degree region. When the sensors report **010** we know the device is in the 135 to 180 degree region.

Generating the Gray code shown in Figure 1 begins with the binary equivalent of the numbers 0 to 2^k-1 (as shown in *Figure 3*). Now transform each of these numbers by replacing each bit (except the leftmost) by its sum (modulo 2) with the preceding (untransformed) bit. For example, consider the transformation of the number **110**. The leftmost bit is unchanged and remains **1**. The next bit is replaced by the sum of **1** and **1** (modulo 2), which is **0**. The rightmost bit is replaced by the sum of **1** and **0** (modulo 2), which is **1**. This yields the value **101**, as shown by the corresponding row of the sequence shown in *Figure 1*.

In this problem you are to display the particular k-bit binary value in a Gray code sequence

corresponding to a particular angle *A*. You will be provided with the values of *k* and *A* in the input data. *k* will be no larger than 32. *A* will be an integer in the range 0 to 360, and will never correspond to an angle at which the sensors change. For the example given above, *A* will never be a multiple of 45. Assume that angles increase in the order shown in the example above. The Gray code value containing only zeroes corresponds to the region with the smallest angles.

Input

There will be multiple test cases in the input, each containing integer values for *k* and *A*, in that order. A pair of zeroes will follow the last test case.

Output

For each input case, display the case number (these are sequential, starting with 1) and the appropriate *k*-bit binary value. Separate each bit in the output with a single blank. The output should appear in the same format as shown in the example below.

Sample Input

3 10

3 350

3 91

0 0

Expected Output

1: 0 0 0

2: 1 0 0

3: 0 1 1

1999-2000 ACM North Central Regional Programming Contest

Problem 2 — Cycles, Period.

A scientist is making observations (recorded as integer values) of a process that is believed to be cyclic. That is, if the period of the process is *p*, then the process values observed at times 0, *p*, 2*p*, … are all the same, as are those values observed at times 1, *p*+1, 2*p*+1, …, and so forth. The scientist also expects that the values that occur in a single cycle are all unique. That is, if the value observed at time *t* is s_t, then for all *i* and *j* such that $0 < i < p$, $0 < j < p$, we expect that $s_i \neq s_j$ for $i \neq j$.

Unfortunately, the scientist cannot continuously observe the process, so observations are made at somewhat random times. In a log, the scientist records observations as a sequence of pairs of numbers, *t* and s, where *t* is the integer elapsed time (since the first observation), and *s* is the value obtained at that time. The log is maintained in order of increasing time of the observations.

For example, suppose a particular process goes through a cycle with values of 1, 2, 3, 4 and 5. If we assume the first observed value (at time $t = 0$) is 1, then the scientist's log might contain the following pairs of integers: (0, 1), (3, 4), (6, 2), (8, 4), (12, 3), (21, 2), (25, 1).

The scientist wants to know if the process is cyclic, and if it is, the period *p* of the process. Your aid has been enlisted. You are to write a program that uses the data from the log as input, and then reports the cycle period for the process if possible. If no cycle can be determined (either because insufficient data is provided, or the data doesn't support the existence of a cycle), then an appropriate message is to be displayed.

The period you are to report is the longest that the data supports. For the data to support the assumption that the process is cyclic, the following conditions must hold.

- At least one observed value must occur at least two times. In this case the difference between the times associated with the occurrences must be an integer multiple of the cycle period.
- The difference between the times associated with repeated occurrences of the same data value must be a multiple of the cycle period.

If no observed process values are repeated in the log, then you are to assume the process is not cyclic and report that fact.

Input

The input data will contain multiple test cases. Each test case will consist of an arbitrary number of observations reported as pairs of integers, each pair containing the time of observation and the process value. Process values will always be greater than zero. The input for each test case will be terminated with a pair of zeroes, which are not to be treated as data. An additional pair of zeroes will follow the last test case, effectively representing a test case with no observations, which is not to be processed.

Output

For each test case, determine if the given log data indicates the process is cyclic. If so, report the longest possible period supported by the data in a form similar to that shown below. If the log data does not support identification of the process as cyclic, then report that fact, again in a form similar to that shown below.

Sample Input

0 1 3 4 6 2 8 4 12 3 21 2 25 1 0 0

0 1 10 1 20 1 0 0

0 1 10 2 0 0

0 0

Expected Output

Case 1: cycle time = 5.

Case 2: cycle time = 10.

Case 3: no cycle time can be determined.

1999-2000 ACM North Central Regional Programming Contest

Problem 3 — Wild Cards

The ability to use *wild card* characters to select file names is common in many operating system command languages. The mechanism used is as follows:

1. A set of file names *S* is given from which file names will be selected. This is frequently the set of names of files in the current directory.
2. A pattern *P* is given that may include zero or more wild card characters.
3. The pattern *P* is tested against each file name in *S*, with those that match the pattern being selected.

Patterns can be formed in different ways depending on the command language. In MS-DOS, for example, the wild card character '?' will match any single character, and the wild card character '*' will match any sequence of adjacent characters of length 0 or more. Any other non-blank character will only match itself. For example, the pattern A?*?X will match file names ABCX, ABCDX, and ABCDEX, but will not match AX or ABX. The pattern *X* will match any file name that includes an X anywhere, while the pattern ?X* will match any file name with at least two characters that has an X in the second position. Note that the pattern X**X will match what X*X matches.

In this problem you are to determine which file names in a given set match each of a sequence of patterns that may include the * and ? wild card characters. These wild card characters are to be interpreted as described in the preceding paragraph.

Input

The input will begin with a set of file names, one per line starting in the first column and ending with the end of line. File names will include only upper and lower case alphabetic characters and decimal digits. The case of the alphabetic characters is significant (that is, 'A' is not the same as 'a'). A line containing a single dollar sign ('$') follows the last line of the set of file names.

Following the file names will appear a sequence of patterns, one per line starting in the first column and ending with the end of line. Patterns will include only upper and lower case alphabetic characters, digits, asterisks ('*') and question marks ('?'). A line containing a single dollar sign ('$') indicates the end of the sequence of patterns.

Each pattern and filename will contain between 1 and 20 characters.

Output

For each pattern, determine which file names in the set are matched. Display the pattern and the matching file names in a format similar to that shown the sample output below. Note that the matching file names are to be displayed in the order they appeared in the input.

Sample Input

AX

ABX

ABCX

ABCDX

ABCDEX

ABCDEF

AABBCC

XXXXYXXXXX

$

ABCDEF

ABCDE

A?*?X

X

?X*

X*X

XX**

A?C*

A*

AA*

X*Y*X

$

Expected Output

Case 1. Pattern ABCDEF matches...

ABCDEF

Case 2. Pattern ABCDE matches...

Case 3. Pattern A?*?X matches...

ABCX

ABCDX

ABCDEX

Case 4. Pattern *X* matches...

AX

ABX

ABCX

ABCDX

ABCDEX

XXXXYXXXXX

Case 5. Pattern ?X* matches...

AX

XXXXYXXXXX

Case 6. Pattern X*X matches...

XXXXYXXXXX

Case 7. Pattern XX matches...**

XXXXYXXXXX

Case 8. Pattern A?C* matches...

ABCX

ABCDX

ABCDEX

ABCDEF

Case 9. Pattern A* matches...

AX

ABX

ABCX

ABCDX

ABCDEX

ABCDEF

AABBCC

Case 10. Pattern AA* matches...

AABBCC

Case 11. Pattern *X*Y*X* matches...

XXXXYXXXXX

1999-2000 ACM North Central Regional Programming Contest

Problem 4 — Clock Angles

The angle between the hour and minute hands of an analog clock is always between 0 and 180 degrees, inclusive. For example, at 12:00 the angle between the hands is 0 degrees. At 6:00 the angle is 180 degrees, and at 3:00 the angle is 90 degrees. In this problem you are to find the angle between the hands of a clock for an arbitrary time between 12:00 and 11:59.

Input

The input data will contain multiple test cases. Each test case will consist of two numbers. The first number specifies the hour, and will be greater than 0 and less than or equal to 12. The second number specifies the number of minutes, and will be in the range 0 to 59. Two numbers, each of which is 0, will follow the last test case.

Output

For each test case, display the time in the usual form and the smallest angle formed by the hands, accurate to one fractional digit. The output should be similar to that shown in the sample below.

Sample Input

12 0

12 30

6 0

3 0

0 0

Expected Output

At 12:00 the angle is 0.0 degrees.

At 12:30 the angle is 165.0 degrees.

At 6:00 the angle is 180.0 degrees.

At 3:00 the angle is 90.0 degrees.

1999-2000 ACM North Central Regional Programming Contest

Problem 5 — Going Down!

Every positive integer *n* can be expressed as the sum of one or more non-increasing sequences of one or more positive integers, each less than or equal to *n*. For example, the integer 4 can be expressed as 1 + 1 + 1 + 1, or 2 + 1 + 1, or 2 + 2, or 3 + 1, or 4. In this problem you are determine the number of sequences that meet this requirement for a particular values of *n*, and also have exactly *k* terms. For example, with *n* = 4 and *k* = 2, there are two sequences that meet the criteria: 2 + 2 and 3 + 1. Note that 1 + 3 does not satisfy the requirements of the problem, as the sequence in that case is not non-increasing.

Input

There will be multiple test cases. For each test case the input will contain values for *n* and *k*. A pair of zeroes will follow the last test case. The value of *n* will always be at least 1 and no larger than 100. The value of *k* will be at least 1 and no larger than *n*.

Output

For each test case, print a line containing the case number; these are numbered sequentially, starting with 1. Then display the number of non-increasing sequences including exactly *k* integers with sums equal to *n*, including the values of *k* and *n*. The grammar of your response should be correct! The example output shown below illustrates an acceptable format.

Sample Input

5 2

5 1

2 2

0 0

Expected Output

Case 1: 2 non-increasing sequences with 2 integers have 5 as their sum.

Case 2: One non-increasing sequence with one integer has 5 as its sum.

Case 3: One non-increasing sequence with 2 integers has 2 as its sum.

1999-2000 ACM North Central Regional Programming Contest

Problem 6 — Hamming It Up

If we wish to devise an information encoding scheme that permits us to detect errors, we need to add redundant information. Consider, for example, that we want to send one of four possible messages over a communication channel. Digitally, the most obvious way to do this is to associate the four binary *codewords* **00**, **01**, **10**, and **11** with the four messages, and then send the appropriate codeword over the channel.

But what happens if noise or electrical problems during transmission cause one of the bits in a codeword to be changed? For example, suppose we sent the codeword **00** but it was received as **01**. Is there any way the receiver of the codeword **01** can detect that it was really supposed to be **00**?

R. W. Hamming, almost half a century ago, studied this problem. One measure of the ability of a set of codewords to support reliable communication is the *Hamming distance* of the set. The Hamming distance between any two codewords is the number of bit positions in which the codewords differ. For example, the codewords **10010** and **11000** differ in two bit positions (the second and fourth bits from the left), so the Hamming distance between them is 2. The Hamming distance of a set of codewords is just the minimum Hamming distance between any two codewords in the set. The Hamming distance for the set of codewords including **10010**, **11000**, and **11111** set would be 2 (since the Hamming distance between **10010** and **11111** is 3, and the Hamming distance between **11000** and **11111** is also 3). The Hamming distance of our original set of two-bit codewords **00**, **01**, **10**, and **11** is 1, insufficient to even detect a 1-bit error.

In this problem you are to determine the Hamming distance for each of several sets of codewords. Each codeword in a set will be assumed to be 32 bits long, but will be represented in the input data as a signed decimal integer. For example, the set of codewords **00**, **01**, **10**, and **11** (assuming a suitable number of leading zero bits) would be specified as 0, 1, 2 and 3 respectively. Likewise, the codewords **10010**, **11000**, and **11111** would be specified as 18, 24, and 31 respectively. If an input decimal integer is negative, then the corresponding codeword is the binary two's complement representation of the number. There will be no more than 20 codewords in any set.

Input

There will be multiple test cases in the input. Each test case begins with an integer n ($2 \pounds n \pounds 20$) specifying the number of codewords in the set. This is followed by n signed decimal integers giving the value of the codewords. A single 0 will follow the last test case.

Output

For each input case, display the case number (these are sequential, starting with 1) and the Hamming distance for the set of codewords in that test case. Display the results in a format similar to that shown below.

Sample Input

4 0 1 2 3

3 18 24 31

2 1 –1

2 0 -1

0

Expected Output

Case 1. Hamming distance = 1.

Case 2. Hamming distance = 2.

Case 3. Hamming distance = 31.

Case 4. Hamming distance = 32.