1995 Spring Denison Computer Programming Contest
Problem 1
You "Month" Remember This . . .

Here's an old chestnut: Given a month and a year, print the calendar. Recall that a year has 365 days except for leap years. Leap years occur on every year that is divisible by 4, unless that year is also divisible by 100 (in which case it is not a leap year), unless that year is also divisible by 400 (in which case it is a leap year). For example, the years 1992, 1904 and 2000 are leap years but the years 1900, 1963 and 2100 are not. We will not ask for a calendar before the year 1753, since the English-speaking world adopted this method of making calendars in September 1752 (and lost 11 days in the process). Handy programming hint: Today is Saturday, 18 February, 1995.

So, your input will be a series of lines. Each line will contain two integers m and y. The integer m represents the month ($1 \leq m \leq 12$) and the integer y represents the year ($1752 < y \leq 10,000$). For each month and year, you should print the correct calendar for that month. The days of the month should appear right-justified in a field of width 3, the calendar should appear in the usual format of a seven-day week, Sunday first, and the first row of your output should be headings indicating the day of the week (`S, M, T, W, T, F, S`) *centered* above their appropriate columns; see the examples below. Input will end when $m = 0 = y$ and this should generate no output. Each calendar should be separated by exactly one blank line.

Examples

On input:

```
4 1951
2 1904
3 3000
0 0
```

the output should look like:

```
 S  M  T  W  T  F  S
  1  2  3  4  5  6  7
  8  9 10 11 12 13 14
 15 16 17 18 19 20 21
 22 23 24 25 26 27 28
 29 30

 S  M  T  W  T  F  S
     1  2  3  4  5  6
  7  8  9 10 11 12 13
 14 15 16 17 18 19 20
 21 22 23 24 25 26 27
 28 29

 S  M  T  W  T  F  S
                    1
  2  3  4  5  6  7  8
  8 10 11 12 13 14 15
 16 17 18 19 20 21 22
 23 24 25 26 27 28 29
 30 31
```

1995 Spring Denison Computer Programming Contest
Problem 2
Farey Interesting

A *Farey sequence of order N* is all reduced fractions between 0 and 1 (inclusive) whose denominators are N or less, arranged in increasing order. For example, the Farey sequence of order 6 is:

$$\frac{0}{1}, \frac{1}{6}, \frac{1}{5}, \frac{1}{4}, \frac{1}{3}, \frac{2}{5}, \frac{1}{2}, \frac{3}{5}, \frac{2}{3}, \frac{3}{4}, \frac{4}{5}, \frac{5}{6}, \frac{1}{1}$$

There is an interesting algorithm to build a Farey sequence known as the Stern-Brocot algorithm: To get the Farey sequence of order N you make $N-1$ passes over the sequence, adding reduced fractions between existing ones. You start with the Farey sequence of order 1 (which has precisely two fractions: $\frac{0}{1}, \frac{1}{1}$), after the first pass you get the Farey sequence of order 2, after the second pass you get the Farey sequence of order 3, and so on. It works as follows: If $\frac{a}{b}$ and $\frac{c}{d}$ are adjacent fractions in the previous Farey sequence then add the fraction $\frac{a+c}{b+d}$ between the two. One can prove that it is reduced and in the correct order! If in this process the denominator of the new fraction, $b+d$ is larger than N, and so should not be included in the Farey sequence of order N, simply exclude it from the list. We demonstrate with the first three passes:

Original sequence: $\frac{0}{1}, \frac{1}{1}$.

After one pass: $\frac{0}{1}, \frac{1}{2}, \frac{1}{1}$.

After two passes: $\frac{0}{1}, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}, \frac{1}{1}$.

After three passes: $\frac{0}{1}, \frac{1}{4}, \frac{1}{3}, \frac{2}{5}, \frac{1}{2}, \frac{3}{5}, \frac{2}{3}, \frac{3}{4}, \frac{1}{1}$.

Neat, huh? These sequences can get quite long. In this problem you will be given three positive integers: N, a and b and are to determine where in the Farey sequence of order N the reduced fraction $\frac{a}{b}$ appears and print this fraction along with the three fractions immediately preceding it and the three fractions immediately following it. You may assume that $N \leq 200$, that $a < b \leq N$ and a and b are relatively prime and that they don't appear within 3 positions of the beginning or end or the sequence. These numbers will appear on one line in the order given above and you should continue to process input until $N = 0$. For each input line, you should produce three output lines. The first line should give the position of the fraction $\frac{a}{b}$ in the Farey sequence of order N using the format given in the examples below. (This wording should be followed exactly.) And the second line should be seven fractions (again using the format given in the examples) each separated by exactly one space. The third line should be blank.

Examples

On input:

```
100 3 10
73 17 50
6 1 2
0 0 0
```

the output should look like:

```
This fraction appears in position 915 of the sequence.
23/77 26/87 29/97 3/10 28/93 25/83 22/73

This fraction appears in position 562 of the sequence.
20/59 19/56 18/53 17/50 16/47 15/44 14/41

This fraction appears in position 7 of the sequence.
1/4 1/3 2/5 1/2 3/5 2/3 3/4
```

1995 Spring Denison Computer Programming Contest
Problem 3
Oh, oh, oh . . . it's Magic

A magic square of size n is an $n \times n$ grid of numbers such that the numbers in each row, each column and the two diagonals all sum to the same value. If that value is S, we say that *the magic square totals to* S. When n is odd, there is a method to insert any n^2 values $p_1, p_2, \ldots, p_{n^2}$ of the form $a, a+b, a+2b, \ldots, a+(n^2-1)b$ into the grid to form a magic square ($b > 0$). We first assume the locations in the square are numbered (i, j), where i is the row number and j is the column number. To form a magic square, we proceed as follows:

1. Place p_1 in the center location in the bottom row.
2. If p_{m-1} is in location (i, j) then place p_m in location $(i+1, j+1)$ if that location is empty.
3. Otherwise, place p_m in location $(i-1, j)$.

If the above directions cause p_m to be placed off the grid, simply "wrap around": if p_m falls off the bottom, place it in the top row in the same column; if p_m falls off the top, place it in the bottom row in the same column; if p_m falls off the right, place it in left-most column in the same row. This is illustrated in the example shown below, where $n = 5$, $a = 1$, $b = 2$. We show 3 snapshots of the process: after the first 6 values have been put in; after the first 11 value have been put in; after all 25 elements have been put in.

			3	
				5
7	11			
	9			
		1		

first 6 elements

21			3	17
19				5
7	11			
	9	13		
		1	15	

first 11 elements

21	35	49	3	17
19	23	37	41	5
7	11	25	39	43
45	9	13	27	31
33	47	1	15	29

finished square

The finished 5×5 magic square totals to 125. The object of this programming program is, given a value S, count all possible 3×3, 5×5 and 7×7 magic squares made up of *positive integers* that total to S. We are only interested in magic squares constructed according to the rules above.

Input will consist of one problem instance, the value S, $S \leq 15,000$. If there is at least one 3×3 square, you should print the one with the smallest starting element p_1; this applies to the other two sizes as well. Each $n \times n$ square should be printed on n lines, each line containing n values output using a field width of 5. A blank line should follow each square. Afterwards you should output three lines of the form

```
There are <m> <n>x<n> magic squares which total to <S>
```

where <n> is 3, then 5, then 7, <m> is the corresponding number of squares totaling to <S>, and <S> is the input value.

Example

On input:

```
10017
```

the output should look like:

```
 2505 6675  837
 1671 3339 5007
 5841    3 4173

 1254 1785 2316 2847   74  605 1136
 1195 1313 1844 2375 2493  133  664
  723  841 1372 1903 2434 2552  192
  251  782  900 1431 1962 2080 2611
 2670  310  428  959 1490 2021 2139
 2198 2729  369  487 1018 1549 1667
 1726 2257 2788   15  546 1077 1608

There are 834 3x3 magic squares which total to 10017
There are 0 5x5 magic squares which total to 10017
There are 59 7x7 magic squares which total to 10017
```

On input:

```
14
```

the output should look like:

```
There are 0 3x3 magic squares which total to 14
There are 0 5x5 magic squares which total to 14
There are 0 7x7 magic squares which total to 14
```

1995 Spring Denison Computer Programming Contest
Problem 4
Radar...oh Really?

One of the natural wonders of Moldavia is (of course) Grand Lake Moldavia located in the center of the country. This lake is unique in many ways. For one, it is perfectly square. For another, it has a rather unusual way to monitor traffic on the lake. The lake is divided into an 8×8 grid and a radar station is located at the end of each row and column. Moldavian radar is unusual in that it does not report distances; instead, each station reports the total number of grid squares in their row or column that are currently occupied by boats. An example is shown below, where a ■ in a grid indicates the presence of a boat (or part of a boat). The totals reported by the 16 radar installations are shown on the right and bottom sides:

■							■	2
			■	■	■		■	4
	■							1
	■		■					2
	■				■	■		3
								0
■								1
		■	■			■		3
2	3	1	3	1	2	2	2	

The diagram above also shows the complete Moldavian fleet: two 1×3 ships, three 1×2 ships and four 1×1 ships. Each ship will only be located on the grid in a vertical or horizontal position, and no ship can abut any other ship, even diagonally (by order of the Moldavian Fleet Port Authority). Thus, the boat located in the upper-left hand corner could not be located one position down or two positions to the right. For this problem, you will be given a set of 16 radar readings and you will output the total number of different ways the 9 Moldavian ships could be located in Grand Lake Moldavia. IMPORTANT POINT: ships of the same size are considered identical; switching two (or more) identical ships in one layout does NOT produce a new layout.

Input will consist of multiple problem instances. Each instance will consist of two lines: the first will contain the eight integers corresponding to the 8 radars on the right side of the lake, and the second will contain eight integers corresponding to the 8 radars on the bottom. End-of-file will terminate input

Output should consist of one line per problem instance, containing the total number of layouts of ships for that problem.

Example

On input:

```
2 4 1 2 3 0 1 3
2 3 1 3 1 2 2 2
4 3 2 0 0 1 2 4
3 3 2 2 2 1 1 2
5 1 1 2 1 1 2 3
3 1 1 6 1 0 1 3
```

the output should look like:

```
183
2
42
```

1995 Spring Denison Computer Programming Contest
Problem 5
Garbage In, Garbage Out

Recycling glass requires that the glass be separated by color into one of three categories: brown glass, green glass and clear glass. In this problem you will be given three recycling bins, each containing a specified number of brown, green and clear bottles. In order to be recycled, the bottles will need to be moved so that each bin contains bottles of only one color.

The problem is to assign a color to each bin which minimizes the number of bottles that are moved. For the purposes of this problem, each bin has sufficient capacity and the only constraint is moving the bottles so that each bin contains bottles of a single color. The maximum number of bottles in all three bins will be ≤ 1000.

The input consists of multiple problem instances. Each instance consists of one line containing 9 integers, separated by one space. The first three integers represent the number of brown, green and clear bottles (respectively) in bin number 1, the second three represent the number of brown, green and clear bottles (respectively) in bin number 2, and the third three represent the number of brown, green and clear bottles (respectively) in bin number 3. For example, the line

```
10 15 20 30 12 8 15 8 31
```

indicates that there are 10 brown, 15 green and 20 clear bottles in bin 1, 30 brown, 12 green and 8 clear bottles in bin 2, and 15 brown, 8 green and 31 clear bottles in bin 3. End-of-file terminates input.

For each problem instance there will be one line of output indicating the bin color assignment which minimizes the number of bottles moved. You should also print the minimum number of bottles moved. The output should consist of a string of the three upper case characters `'G'`, `'B'`, `'C'` representing the color in each bin: the first character is the color of bottles in the first bin, and so on. There should be no spaces between these three characters. They should be followed by one space and an integer indicating the total number of bottles moved. If more than one order of brown, green and clear bins yields the minimum number of movements then the alphabetically first string representing a minimal configuration should be printed.

Example

On input:

```
1 2 3 4 5 6 7 8 9
5 10 5 20 10 5 10 20 10
```

the output should look like:

```
BCG 30
CBG 50
```