The 33rd Annual ACM International Collegiate Programming Contest

Sponsored by IBM

Arab and North Africa Eleventh Regional Contest

Arab Academy for Science and Technology
Alexandria, Egypt
November 2008

The problem set is made of 14 numbered pages

[A] Tobo or not Tobo

Program:	`tobo.(c|cpp|java)`
Input:	`tobo.in`
Balloon Color:	`Orange`

Description

The game of Tobo is played on a plastic board designed into a 3x3 grid with cells numbered from 1 to 9 as shown in figure (a). The grid has four dials (labeled "A" to "D" in the figure.) Each dial can be rotated in 90 degrees increment in either direction. Rotating a dial causes the four cells currently adjacent to it to rotate along. For example, figure (b) shows the Tobo after rotating dial "A" once in a clockwise direction. Figure (c) shows the Tobo in figure (b) after rotating dial "D" once in a counter-clockwise direction.

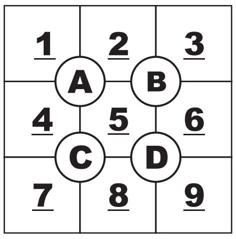


Figure (a)

Kids love to challenge each other playing the Tobo. Starting with the arrangement shown in figure (a), (which we'll call *the standard arrangement,*) one kid would randomly rotate the dials, `X` number of times, in order to 'shuffle" the board. Another kid then tries to bring the board back to its standard arrangement, taking no more than `X` rotations to do so. The less rotations are needed to restore it, the better. This is where you see a business opportunity. You would like to sell these kids a program to advise them on the minimum number of steps needed to bring a Tobo back to its standard arrangement.

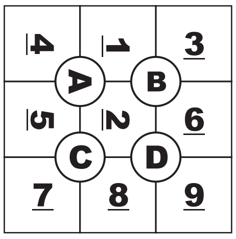


Figure (b)

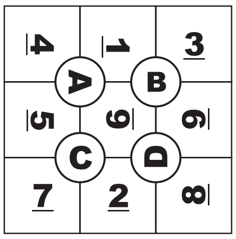


Figure (c)

Input Format

Your program will be tested on one or more test cases. Each test case is specified on a line by itself. Each line is made of 10 decimal digits. Let's call the first digit `Y`. The remaining 9 digits are non-zeros and describe the current arrangement of the Tobo in a row-major top-down, left-to-right ordering. The first sample case corresponds to figure (c).
The last line of the input file is a sequence of 10 zeros.

Output Format

For each test case, print the result using the following format:

`k.␣R`

where `k` is the test case number (starting at 1,) ␣ is a single space, and `R` is the minimum number of rotations needed to bring the Tobo back to its standard arrangement. If this can't be done in `Y` dials or less, then `R=-1`.

Sample Input/Output

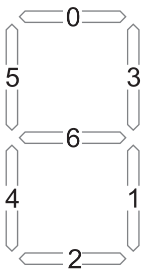
tobo.in
```
3413569728
1165432789
0000000000
```

OUTPUT
```
1. 2
2. -1
```

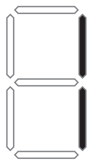
[B] Adding Sevens

Program:	`seven.(c|cpp|java)`
Input:	`seven.in`
Balloon Color:	`Red`

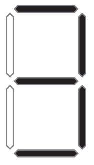
Description



A seven segment display, similar to the one shown on the right, is composed of seven light-emitting elements. Individually on or off, they can be combined to produce 127 different combinations, including the ten Arabic numerals. The figure on the next page illustrates how the ten numerals are displayed. 7-seg displays (as they're often abbreviated) are widely used in digital clocks, electronic meters, and calculators. A 7-seg has seven connectors, one for each element, (plus few more connectors for other electrical purposes.) Each element can be turned on by sending an electric current through its pin. Each of the seven pins is viewed by programmers as a single bit in a 7-bit number, as they are more comfortable dealing with bits rather than electrical signals. The figure on the right shows the bit assignment for a typical 7-seg, bit 0 being the right-most bit. For example, in order to display the digit 1, the programmer knows that only bits 1 and 3 need to be on, i.e. the 7-bit binary number to display digit 1 is "0001010", or 10 in decimal. Let's call the decimal number for displaying a digit, its *display code,* or just *code* for short. Since a 7-seg displays 127 different configurations, display codes are normally written using 3 decimal places with leading zeros if necessary, i.e. the display code for digit 1 is written as 010.


0001010
code: 010

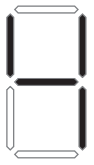
In a 9-digit calculator, 9 7-seg displays are stacked next to each other, and are all controlled by a single controller. The controller is sent a sequence of $3n$ digits, representing n display codes, where $0 < n < 10$. If $n < 9$, the number is right justified and leading zeros are automatically displayed. For example, the display code for 13 is 010079 while for 144 it is 010106106


1001111
code: 079

Write a program that reads the display codes of two numbers, and prints the display code of their sum.

Input Format

Your program will be tested on one or more test cases. Each test case is specified on a single line in the form of `A+B=` where both `A` and `B` are display codes for decimal numbers a and b respectively where $0 < a, b < a + b < 1,000,000,000$. The last line of the input file is the word `"BYE"` (without the double quotes.)


1101010
code: 106

Output Format

For each test case, print `A+B=C` where `C` is the display code for $a + b$.

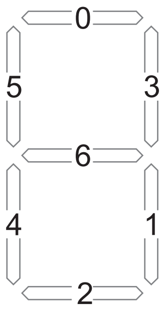
Sample Input/Output

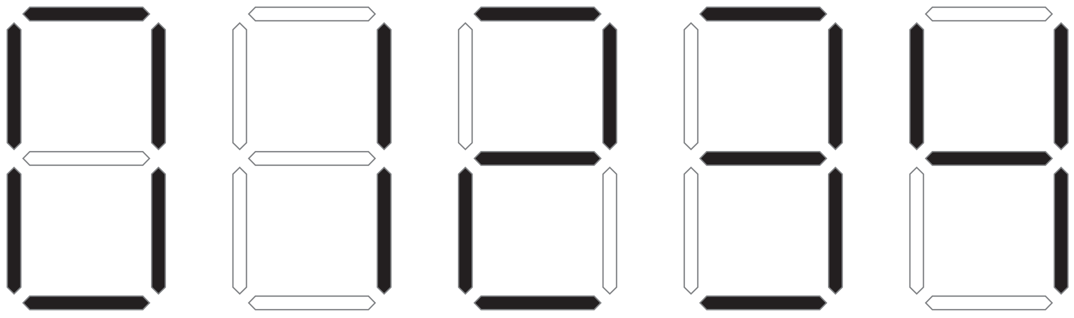
seven.in

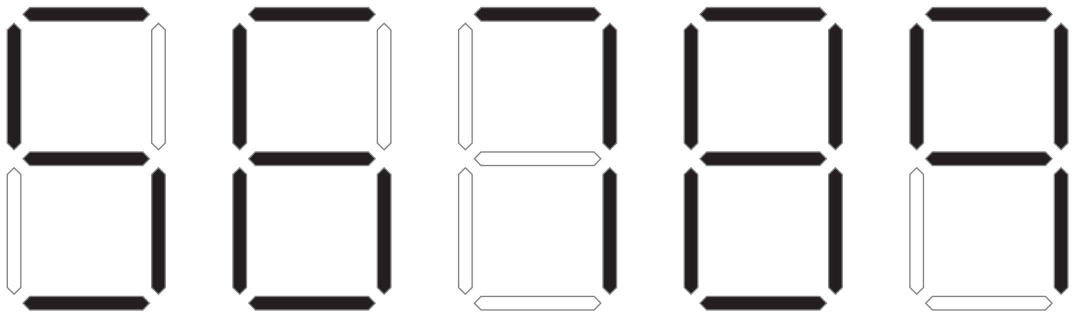
```
010079010+010079=
106010+010=
BYE
```

OUTPUT

```
010079010+010079=010106106
106010+010=106093
```


0
5
3
6
4
1
2





[C] Match Maker

Program:	`match.(c|cpp|java)`
Input:	`match.in`
Balloon Color:	`Gold`

Description

In Computer Science, *pattern matching* is the act of checking if a certain sequence conforms (matches) a given pattern. Patterns are usually specified using a language based on *regular expression.* In this problem, we'll use a simple regular expression to express patterns on sequences of decimal digits. A pattern is a sequence of one or more decimal digits `'0'`... `'9'`, asterisks `'*'`, and hash signs `'#'`. A `'*'` denotes a sequence of an even number of digits, whereas a `'#'` denotes a sequence of an odd number of digits. For example, the pattern `"129"` only matches the sequence 129. The pattern `"1*3"` matches all sequences beginning with 1, ending with 3, and having an even number of decimal digits between the first and last digits. As another example, the pattern `"#55"` matches the sequences 155, 12355, 1234555, but none of the sequences 55, 1255, 123455. Your task is to write a program to find if a given sequence matches a given pattern.

Input Format

Your program will be tested on one or more data sets. Each data set contains a single pattern and one or more sequences to match. The first line of each data set specifies the pattern, and the remaining lines specify the sequences to match against that pattern. The end of a data set (except the last) is identified by the word `"END"` (without the double quotes.) The end of the last data set is identified by the word `"QUIT"`. All lines are 100,000 characters long or shorter.

Output Format

```
k.s.␣result
```

Where `k` is the test case number (starting at one,) and `s` is the sequence number (starting at one within each test case,) and result is either the word `"match"` if the given string matches the pattern, or the word `"not"` if it doesn't.

Sample Input/Output

match.in

```
129
1299
129
1129
END
1*3
123
1223
END
#55
155
12355
55
1255
QUIT
```

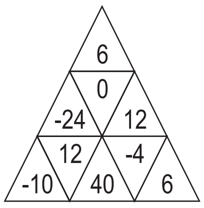
OUTPUT

```
1.1. not
1.2. match
1.3. not
2.1. not
2.2. match
3.1. match
3.2. match
3.3. not
3.4. not
```

[D] Adding up Triangles

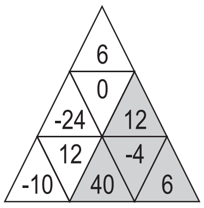
Program:	`tri.(c|cpp|java)`
Input:	`tri.in`
Balloon Color:	`Green`

Description



Take a look at the triangle on the right. It is made of 9 (unit) triangles arranged in three rows (`N=3`). Needless to say, a unit triangle is a triangle with `N=1`. If you study the figure for few seconds, you'll realize that you can find 13 different triangles (which we'll call sub-triangles.) Of these 13 sub-triangles we have: Nine unit triangle; three with `N=2`, and one with `N=3`. The following table lists the number of sub-triangles in arrangements with $N < 5$.

# of Rows:	$N = 1$	$N = 2$	$N = 3$	$N = 4$
# of Sub-triangles:	1	5	13	27



Let's define the value of a unit triangle to be the integer value written in that triangle. In general, the value of a triangle is the sum of values in all its unit triangles. The figure on the right is the same as the one above but with the sub-triangle having the largest value being highlighted. Write a program to determine the sub-triangle with the largest value.

Input Format

Your program will be tested on one or more test cases. Each test case is specified in a single line made of integers (separated by spaces.) The first integer is the number of rows in the test case, and the remaining integers are the values of the unit triangles specified in a top-down, left-to-right order. (the first test case in the example below is the same as the one in the figure.) The last line of the input file contains the number 0 (which is not part of the test cases.)

The maximum number of rows is 400. The absolute value of a unit triangle is less than 1000.

Output Format

For each test case, print the result using the following format:

```
k.␣V
```

where `k` is the test case number (starting at 1,) ␣ is a single space, and `V` is the maximum value of a sub-triangle in that test case.

Sample Input/Output

tri.in
```
3 6 -24 0 12 -10 12 40 -4 6
4 1 1 -1 1 1 -1 1 -1 1 1 -1 1 -1 1 -1 1
0
```

OUTPUT
```
1. 54
2. 4
```

[E] Relax! It's just a game

Program:	`game.(c|cpp|java)`
Input:	`game.in`
Balloon Color:	`Blue`

You: What's the score? Did I miss much?

Me: It's 2-1 for elAhli and the second half just started. The first half was quite boring.

You: Who scored first? elAhli or ezZamalek?

Me: What difference does it make?

You: Big difference! I can predict the outcome of the match if I knew the order of which goals were scored in the first half.

Me: What do you mean?

You: It's 2-1 for elAhli, right? One of three things could have happened: elAhli scored two goals then ezZamalek scored; Or, elAhli scored its first goal, then ezZamalek, then elAhli again; Or, ezZamalek scored first, then elAhli scored its two goals.

Me: *So?!!* I still don't understand what difference does that make? It's still 2-1 for elAhli! Why don't you just relax and let us continue watching the game in peace.

You: *You don't understand!!* I believe the probability of who'll win depends on the order of how goals were scored. Now I have to predict the outcome for 3 possibilities.

Me: And what if the score was 3-2? What would you have done then?

You: I would have to work for 5 different possibilities. No?

Me: *Of course not!* The number of possibilities isn't always equal to the sum.

You: Can you tell me when will it be equal to the sum?

Me: You're a programmer, why don't you write a program that counts the number of possibilities and compare it to the sum?

You: I don't have the time, I want to watch the match. Besides, I have nine other problems to worry about.

Me: I'll give you a hint. The possibilities will be equal to the sum only if one of the teams scored a certain number of goals.

Input Format

Your program will be tested on one or more test cases. Each test case specifies two natural numbers (`A` and `B`) (separated by one or more spaces) representing the score of the first half. No team will be able to score more than 10 goals. The last line of the input file contains two -1's (which is not part of the test cases.)

Output Format

For each test case where the number of possibilities is equal to the sum, print:

```
A+B=C
```

Where `A` and `B` are as above and `C` is their sum. If the number of possibilities is not equal to the sum, replace the `'='` sign with `'!='` (without the quotes.)

Sample Input/Output

game.in
```
2 1
1 0
-1 -1
```

OUTPUT
```
2+1=3
1+0=1
```

[F] Einbahnstrasse

Program:	`one.(c|cpp|java)`
Input:	`one.in`
Balloon Color:	`Purple`

Description

Einbahnstraße (German for a one-way street) is a street on which vehicles should only move in one direction. One reason for having one-way streets is to facilitate a smoother flow of traffic through crowded areas. This is useful in city centers, especially old cities like Cairo and Damascus. Careful planning guarantees that you can get to any location starting from any point. Nevertheless, drivers must carefully plan their route in order to avoid prolonging their trip due to one-way streets. Experienced drivers know that there are multiple paths to travel between any two locations. Not only that, there might be multiple roads between the same two locations. Knowing the shortest way between any two locations is a must! This is even more important when driving vehicles that are hard to maneuver (garbage trucks, towing trucks, etc.)

You just started a new job at a car-towing company. The company has a number of towing trucks parked at the company's garage. A tow-truck lifts the front or back wheels of a broken car in order to pull it straight back to the company's garage. You receive calls from various parts of the city about broken cars that need to be towed. The cars have to be towed in the same order as you receive the calls. Your job is to advise the tow-truck drivers regarding the shortest way in order to collect all broken cars back in to the company's garage. At the end of the day, you have to report to the management the total distance traveled by the trucks.

Input Format

Your program will be tested on one or more test cases. The first line of each test case specifies three numbers (N, C, and R) separated by one or more spaces. The city has N locations with distinct names, including the company's garage. C is the number of broken cars. R is the number of roads in the city. Note that $0 < N < 100$, $0 \leq C < 1000$, and $R < 10000$. The second line is made of $C + 1$ words, the first being the location of the company's garage, and the rest being the locations of the broken cars. A location is a word made of 10 letters or less. Letter case is significant. After the second line, there will be exactly R lines, each describing a road. A road is described using one of these three formats:

```
A --v-> B
A <-v-- B
A <-v-> B
```

`A` and `B` are names of two different locations, while `v` is a positive integer (not exceeding 1000) denoting the length of the road. The first format specifies a one-way street from location `A` to `B`, the second specifies a one-way street from `B` to `A`, while the last specifies a two-way street between them. `A`, "the arrow", and `B` are separated by one or more spaces. The end of the test cases is specified with a line having three zeros (for N, C, and R.)

Output Format

For each test case, print the total distance traveled using the following format:

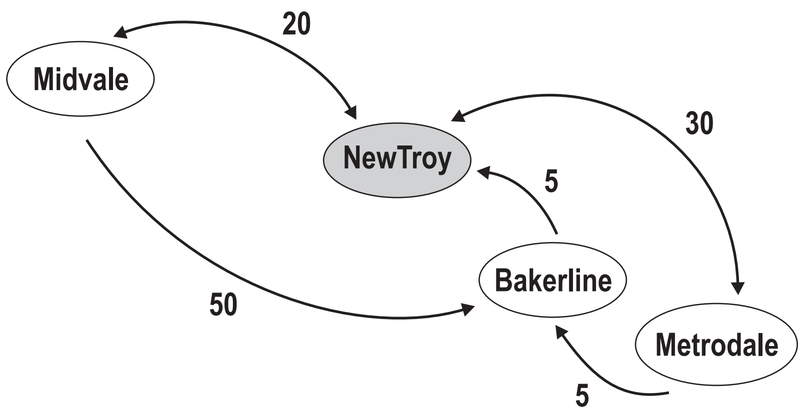
```
k.␣V
```

Where `k` is test case number (starting at 1,) ␣ is a space, and `V` is the result.

Sample Input/Output

one.in
```
4 2 5
NewTroy Midvale Metrodale
NewTroy   <-20-> Midvale
Midvale   --50-> Bakerline
NewTroy    <-5-- Bakerline
Metrodale <-30-> NewTroy
Metrodale  --5-> Bakerline
0 0 0
```

OUTPUT
```
1. 80
```



[G] Think I'll Buy Me a Football Team

Program:	`money.(c|cpp|java)`
Input:	`money.in`
Balloon Color:	`Pink`

Description

Falling Stocks. Bankrupted companies. Banks with no Cash. Seems like the best time to invest: *"Think I'll buy me a football team!"*
No seriously, I think I have the solution to at least the problem of cash in banks. Banks nowadays are all owing each other great amounts of money and no bank has enough cash to pay other banks' debts even though, on paper at least, they should have enough money to do so. Take for example the inter-bank loans shown in figure (a). The graph shows the amounts owed between four banks (`A`...`D`). For example, `A` owes `B` 50M while, at the same time, `B` owes `A` 150M. (It is quite common for two banks to owe each other at the same time.) A total amount of 380M in cash is needed to settle all debts between the banks.
In an attempt to decrease the need for cash, and after studying the example carefully, I concluded that there's a lot of cash being transferred unnecessarily. Take a look:

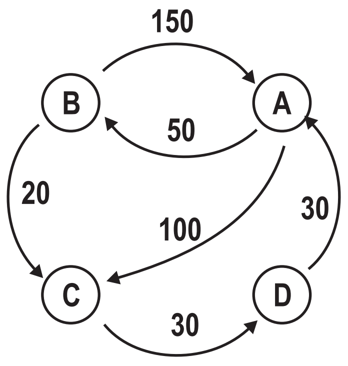


Figure (a)

1. `C` owes `D` the same amount as `D` owes `A`, so we can say that `C` owes `A` an amount of 30M and get `D` out of the picture.
2. But since `A` already owes `C` 100M, we can say that `A` owes `C` an amount of 70M.
3. Similarly, `B` owes `A` 100M only, (since `A` already owes `B` 50M.) This reduces the above graph to the one shown in figure (b) which reduces the needed cash amount to 190M (A reduction of 200M, or 53%.)
4. I can still do better. Rather than `B` paying `A` 100M and `A` paying 70M to `C`, `B` can pay 70M (out of `A`'s 100M) directly to `C`. This reduces the graph to the one shown in figure (c). Banks can settle all their debts with only 120M in cash. A total reduction of 260M or 68%. *Amazing!*

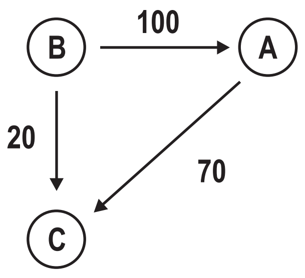


Figure (b)

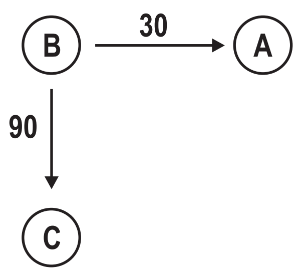


Figure (c)

I have data about inter-bank debts but I can't seem to be able to process it to obtain the minimum amount of cash needed to settle all the debts. Could you please write a program to do that?

Input Format

Your program will be tested on one or more test cases. Each test case is specified on $N + 1$ lines where $N < 1,000$ is the number of banks and is specified on the first line. The remaining N lines specifies the inter-bank debts using an $N \times N$ adjacency matrix (with zero diagonal) specified in row-major order. The ith row specifies the amounts owed by the ith bank. Amounts are separated by one or more spaces. All amounts are less than 1000.

The last line of the input file has a single 0.

Output Format

For each test case, print the result using the following format:

```
k.␣B␣A
```

where `k` is the test case number (starting at 1,) ␣ is a space character, `B` is the amount of cash needed before reduction and `A` is the amount of cash after reduction.

Sample Input/Output

money.in

```
4
  0  50 100   0
150   0  20   0
  0   0   0  30
 30   0   0   0
0
```

OUTPUT

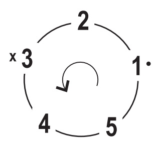
```
1. 380 120
```

[H] Musical Chairs

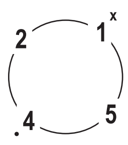
Program:	`chairs.(c|cpp|java)`
Input:	`chairs.in`
Balloon Color:	`Maroon (Brown)`

Description

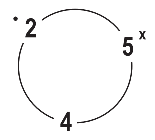
In the traditional game of *Musical Chairs,* $N + 1$ children run around N chairs (placed in a circle) as long as music is playing. The moment the music stops, children run and try to sit on an available chair. The child still standing leaves the game, a chair is removed, and the game continues with N children. The last child to sit is the winner.



In an attempt to create a similar game on these days' game consoles, you modify the game in the following manner: N Children are seated on N chairs arranged around a circle. The chairs are numbered from 1 to N. Your program pre-selects a positive number D. The program starts going in circles counting the children starting with the first chair. Once the count reaches D, that child leaves the game, removing his/her chair. The program starts counting again, beginning with the next chair in the circle. The last child remaining in the circle is the winner.



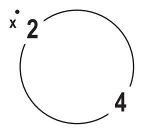
For example, consider the game illustrated in the the adjacent figure for $N = 5$ and $D = 3$. In the figure, the dot indicates where counting starts and $\times$ indicates the child leaving. Starting off, child #3 leaves the game, and counting restarts with child #4. Child #1 is the second child to leave and counting restart with child #2 resulting in child #5 leaving. Child #2 is the last to leave, and child #4 is the winner. Write a program to determine the winning child given both N and D.



Input Format

Your program will be tested on one or more test cases. Each test case specifies two positive integers N and D on a single line, separated by one or more spaces, where $N, D < 1,000,000$.

The last line of the input file contains two 0's and is not part of the test cases.



Output Format

For each test case, write the winner using the following format:

```
N␣D␣W
```

Where N and D are as above, ␣ is a space character, and W is the winner of that game.

Sample Input/Output

chairs.in

```
5 3
7 4
0 0
```

OUTPUT

```
5 3 4
7 4 2
```

[I] I Speak Whales

Program:	`matrix.(c|cpp|java)`
Input:	`matrix.in`
Balloon Color:	`Yellow`

Description

According to Wikipedia, a *Walsh matrix* is a specific square matrix, with dimensions equal to a power of 2, the entries of which are +1 or −1, and the property that the dot product of any two distinct rows (or columns) is zero. Below are the first three Walsh Matrices. (The gray lines are imaginary lines for illustration purpose only.)

$$W_1 = \begin{bmatrix} 1 \end{bmatrix} \qquad W_2 = \left[\begin{array}{r|r} 1 & 1 \\ \hline 1 & -1 \end{array}\right] \qquad W_4 = \left[\begin{array}{rr|rr} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ \hline 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{array}\right]$$

A Walsh Matrix of size 2^{N+1} can be constructed as the "union" of 4 Walsh Matrices of size 2^N arranged such that the lower right matrix is inverted whereas the other 3 matrices are not, i.e.:

$$W_{2^{N+1}} = \left[\begin{array}{c|c} W_{2^N} & W_{2^N} \\ \hline W_{2^N} & -W_{2^N} \end{array}\right]$$

Let's number the rows of a given Walsh Matrix from the top starting with row 0. Similarly, let's number the columns of the matrix from the left starting with column 0. Given the four integers N, R, S, and E, write a program that will construct a Walsh Matrix of size 2^N and will print the sum of all the numbers in row $\#R$ between columns $\#S$ and $\#E$ (inclusive.)

Input Format

Your program will be tested on one or more test cases. Each test case is specified using a single line listing four integers in the following order: N, R, S, and E, where $0 \leq N \leq 60$, $0 \leq R < 2^N$, $0 \leq S \leq E < 2^N$, and $E - S \leq 10,000$.

The last line of the input file has four -1's and is not part of the test cases.

Output Format

For each test case, print the output on a single line.

Sample Input/Output

matrix.in

```
2  1 0 1
48 0 0 47
-1 -1 -1 -1
```

OUTPUT

```
0
48
```

[J] A Day at the Races

Program:	`grand.(c|cpp|java)`
Input:	`grand.in`
Balloon Color:	`Black`

Description

Formula One is the highest class of car racing sports. A typical *Formula One season* consists of a series of races called *"Grands Prix"* which constructors like Ferrari, Renault, etc. and others participate with one or more cars driven by the best drivers in the world. During the season, teams compete in two parallel championships: the *drivers championship* and the *teams championship.*

In the **drivers championship,** drivers compete to achieve the maximum total number of points by the end of the season, the rules of the competition states that the top eight drivers at each Grand Prix receive 10,8,6,5,4,3,2,1 points respectively. In case of points tie, the driver with the highest number of first places leads. If still tied, then the highest second places, and so on till the highest 8th places. If still tied, then drivers are sorted lexicographically by their last and then by their first names.

After each race, the points received by each driver are added to his team's pocket, and at the end of the season the team with the highest number of points wins the **teams championship.** To add excitement to the season, team sponsors are allowed to buy drivers from other teams even within the same season. In case of points tie between teams, teams are sorted lexicographically by their names. In this problem, you are given data of a formula one season and you're asked to process these data according to the rules above to determine both the drivers and teams standings.

Input Format

Your program will be tested on one or more data-sets, each representing a Formula One season. All input lines are 255 characters or less. Studying the sample I/O you'll discover that the first line of each season has an integer N, where $0 < N < 32$ and representing the number of Grands Prix in that season. For each Grand Prix, the name of the Grand Prix appears on a line by itself (maximum length is 64 characters) followed by a table of the first name, last name and team name of the top eight drivers, from 1 to 8, in that Grand Prix. Each of the first and last names is a sequence of printable ASCII characters, no longer than 12 characters, and contains no spaces. Each team name is a sequence of printable ASCII characters, no longer than 18 characters, and may contain spaces (but no leading or trailing spaces.) Each team name is followed by a single period `'.'` which is not part of the name. Trailing white space may follow. A line of three `-`'s follows the listing of each Grand Prix. The last line of the input file contains a single zero.

Output Format

For each data set in the input you must print "`Season k:`" where `k` is the data-set number (starting from 1.) The next line must state "`Drivers Standing:`". On subsequent lines list the drivers standing for that season. For each driver, print their first and last names separated by exactly one space and left justified in a field of width 25, followed by a single space, followed by the total number of points achieved by the driver during the season. The drivers standing should be followed by a blank line.

The next line must state "`Teams Standing:`" On subsequent lines list the teams standing for the that season. For each team, print the team name left justified in a field of width 25, followed by a single space, followed by the total number of points the team has scored during the season. The teams standing should be followed by a blank line.

Sample Input/Output

grand.in

```
2
FORMULA 1 Gran Premio Telefonica de Espana 2006
Pos  Driver                    Team
1    Fernando Alonso           Renault.
2    Michael Schumacher        Ferrari.
3    Giancarlo Fisichella      Renault.
4    Felipe Massa              Ferrari.
5    Kimi Raikkonen            McLaren-Mercedes.
6    Jenson Button             Honda.
7    Rubens Barrichello        Honda.
8    Nick Heidfeld             Sauber-BMW.
---
FORMULA 1 Grand Prix de Monaco 2006
Pos  Driver                    Team
1    Fernando Alonso           Renault.
2    Jaun-Pablo Montoya        McLaren-Mercedes.
3    David Coulthard           RBR-Ferrari.
4    Rubens Barrichello        Honda.
5    Michael Schumacher        Ferrari.
6    Giancarlo Fisichella      Renault.
7    Nick Heidfeld             Sauber-BMW.
8    Ralf Schumacher           Toyota.
---
0
```

OUTPUT

```
Season 1:
Drivers Standing:
Fernando Alonso           20
Michael Schumacher        12
Giancarlo Fisichella      9
Jaun-Pablo Montoya        8
Rubens Barrichello        7
David Coulthard           6
Felipe Massa              5
Kimi Raikkonen            4
Jenson Button             3
Nick Heidfeld             3
Ralf Schumacher           1

Teams Standing:
Renault                   29
Ferrari                   17
McLaren-Mercedes          12
Honda                     10
RBR-Ferrari               6
Sauber-BMW                3
Toyota                    1
```