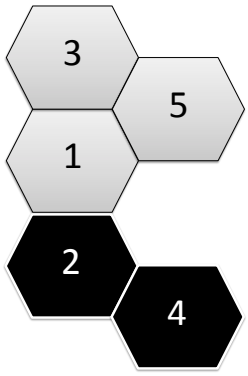
# 2015 Denison Spring Programming Contest
# Granville, Ohio
# 28 February, 2015

Rules:

1. There are **six** problems to be completed in **four hours**.
2. All questions require you to read the test data from standard input and write results to standard output. You cannot use files for input or output. Additional input and output specifications can be found in the General Information Sheet.
3. No whitespace should appear in the output except between printed fields.
4. All whitespace, either in input or output, will consist of exactly one blank character.
5. The allowed programming languages are C, C++ and Java.
6. All programs will be re-compiled prior to testing with the judges' data.
7. Non-standard libraries cannot be used in your solutions. The Standard Template Library (STL) and C++ string libraries are allowed. The standard Java API is available, except for those packages that are deemed dangerous by contestant officials (e.g., that might generate a security violation).
8. The input to all problems will consist of multiple test cases.
9. Programming style is not considered in this contest. You are free to code in whatever style you prefer. Documentation is not required.
10. All communication with the judges will be handled by the $PC^2$ environment.
11. Judges' decisions are to be considered final. No cheating will be tolerated.

# Problem A: Hexagon Crazy

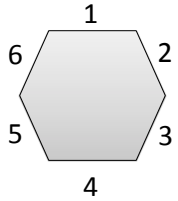
The game `Hive` has two players taking turns to place their hexagonal pieces on a flat surface. The game starts with the first player placing a piece of his or her color on the surface, followed by the second player placing a piece of his or her color aligned along an edge with the first piece. In every subsequent turn the players alternate placing pieces in such a way that their piece must be adjacent to at least one of their own color but not touching any of the opponent's pieces. (The game also allows pieces to subsequently be moved, but we will not concern ourselves with this for this problem.) An example with 5 moves played so far is illustrated below, with the numbers inside the hexagon tiles indicating the order in which they were placed.




Your goal is to count the number of valid positions that the next player has to make his or her move given the sequence of moves played thus far. For example, in the game shown in the above figure, the player with the darker pieces (whose turn it is) can place a piece to the lower left of piece 2, one between pieces 2 and 4, and three more adjacent to piece 4, for a total of five open spots.

## Input

Each input will consist of multiple lines. The first line will be the number of moves made so far ($1 \leq n \leq 100$). The following $(n-1)$ lines will consist of two numbers ($p$ $q$), where $p$ indicates the piece number that the current piece was connected to and $q$ indicates the side of that piece it was placed against. The sides are designated by 1 to 6 in clockwise order starting from the top (see the figure below). You may assume that the given game is legal (no piece is adjacent to a piece of a different color). The input will be terminated by a line with a single 0.



## Output

For each input, you should output the case number followed by the number of legal positions that the next player has to place a tile.

## Sample Input

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

## Sample Output

```
Case 1: 5
```

# Problem B: Meme Team

Every time you and your circle of friends finds a funny meme online you all text it to one another. Unfortunately, your phone carriers charge you (exorbitant!) fees for these picture texts and you would like to figure out a cheap way to disseminate the picture to everyone. You decide to write a program that will determine the cheapest way of doing this for any group of friends and combination of phone carriers.

For example, say that your group of friends has the following carriers

| Friend 1 | Friend 2 | Friend 3 | Friend 4 | Friend 5 |
|---|---|---|---|---|
| Carrier 1 | Carrier 2 | Carrier 1 | Carrier 3 | Carrier 2 |

and the price of sending a picture text is given by the table below (note that the cost is the same in both directions for any two carriers):

| | Carrier 1 | Carrier 2 | Carrier 3 |
|---|---|---|---|
| Carrier 1 | 10 | 15 | 20 |
| Carrier 2 | 15 | 20 | 10 |
| Carrier 3 | 20 | 10 | 20 |

then the cheapest way for the picture to be sent to everyone costs 45 cents. (For example, if Friend 3 finds the picture, she can send it to Friend 1 for 10 cents and Friend 2 for 15 cents; Friend 2 can then send it to Friend 4 for 10 cents; finally, Friend 4 can send it to Friend 5 for 10 cents.)

## Input

Each input will consist of multiple lines. The first line will have the number of friends and the number of carriers $n$ $m$ ($1 \leq n \leq 100$, $1 \leq m \leq n$). The next line will contain $n$ numbers designating the carrier of each friend (i.e., $n$ numbers between 1 and $m$). The following $m$ lines will each contain the cost of sending messages between pairs of carriers. Each cost will be an integer between 1 and 100 (inclusive). You may assume that the data is symmetric—the cost of sending from any carrier A to B is the same as the cost of sending from B to A. A line with 0 0 on it will end the input.

## Output

For each case, you should output the case number followed by the minimum cost of sending the message to everyone.

## Sample Input

```
5 3
1 2 1 3 2
10 15 20
15 20 10
20 10 20
0 0
```

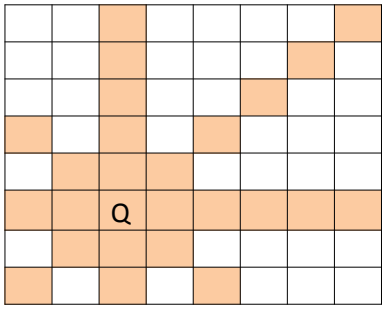
## Sample Output

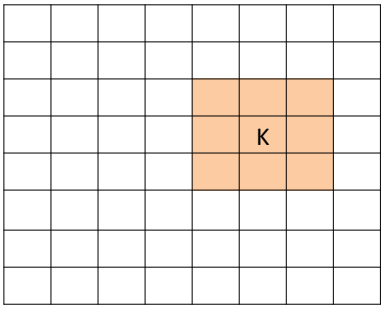
```
Case 1: 45
```

# Problem C: Queens and Kings

Some of you may have heard of the $N$-Queens problem in which you have to determine whether it is possible to place $N$ queens on an $N \times N$ chess board in such a way that no two queens can attack each other. In this problem, you must find a way to place $Q$ queens and $K$ kings on an $N \times N$ chess board in such a way that no piece can attack any other.

Recall that a queen can attack any position along the same row, column, or diagonal from her position. An example of the positions a queen can attack are given in the illustration below. A king can attack any position that is one step away from it, including along the diagonals. See the figure on the right, below, for an example.




For each input, you have to find a way of placing $Q$ queens and $K$ kings on an $N \times N$ board so that no piece attacks another.

## Input

Each input will consist of three numbers on a single line, $N$ $Q$ $K$, where $N$ ($2 \leq N \leq 10$) is the number of rows and columns on the board, $Q$ ($0 \leq Q \leq N$) is the number of queens that must be placed, and $K$ ($0 \leq K \leq N$) is the number of kings that must be placed. The input will be terminated by the line 0 0 0.

## Output

For each input, give the case number followed by a valid board configuration as shown below. Every problem instance given will have a solution. Since there are many possible solutions for a given input, you should output the one that places a queen on the uppermost row possible, breaking ties by putting it on the leftmost column. To break further ties, you should continue to do this with each subsequent queen, followed by the kings. (Note that there should be no spaces after the : on the case number line.)

## Sample Input

```
6 4 3
7 3 7
8 0 5
0 0 0
```

### Sample Output

```
Case 1:
.Q....
...Q..
.....Q
K.....
..Q...
K...K.
Case 2:
.Q.....
...Q...
.....Q.
K.K....
....K..
K......
..K.K.K
Case 3:
K.K.K.K.
........
K.......
........
........
........
........
........
```

## Problem D: Sarah's Sequence Sums

Sarah recently noticed that the number 13 can be written as the sum of the sequences

- (1, 2, 10)
- (1, 3, 9)
- (1, 4, 8)

in which each number is a multiple of the previous one. If you include the boring sequences (13) and $(1, 12)$, there are a total of 5 ways to write 13 as the sum of such sequences. This made Sarah wonder if other numbers also have so many such sequences.

To help Sarah answer this question, you should write a program to count the number of such sequences for any given number $n$. That is, for each input $n$, you should give the number of ways it can be written as a sum of *distinct* positive integers $a_1 + \ldots + a_k$ $(k \geq 1)$ such that the sequence is strictly increasing $(a_1 < a_2 < a_3 < \ldots < a_k)$ and each number is a divisor of the next in the sequence ($a_{i+1}$ is divisible by $a_i$ for $1 \leq i < k$).

### Input

Each input will be a positive integer $1 \leq n \leq 10000$. An input of $n = 0$ indicates the end of problem instances.

### Output

For each input, print the case number and the number of ways the number can be written as the sum of such sequences.

### Sample Input

```
13
10
10000
1
0
```

### Sample Output

```
Case 1: 5
Case 2: 4
Case 3: 910
Case 4: 1
```

## Problem E: Stack Sorting

We consider strings created from the first $n$ lower case letters of the alphabet. For example, when $n = 3$ we can create strings such as "abc", "cab", "bac" and so forth. We all know there will be $n!$ such strings. The alphabetically ordered string ("abc") is called the *standard permutation*. All others are non-standard permutations.

It is possible to "sort" a non-standard permutation into the standard permutation using a stack. At each step, you can either push the next character from the input string onto the stack or pop the stack into the output. For example, consider the string "cab". We start by removing the first character from the input string, the "c", and pushing it onto a stack.

| Step | Description | Input | Stack | Output |
|---|---|---|---|---|
| 0 | Initialize | cab | - | - |
| 1 | Push c | ab | c | - |

We also push the a on the stack.

| Step | Description | Input | Stack | Output |
|---|---|---|---|---|
| 0 | Initialize | cab | - | - |
| 1 | Push c | ab | c | - |
| 2 | Push a | b | ac | - |

Now we pop the a from the stack and add it to the output string.

| Step | Description | Input | Stack | Output |
|---|---|---|---|---|
| 0 | Initialize | cab | - | - |
| 1 | Push c | ab | c | - |
| 2 | Push a | b | ac | - |
| 3 | Pop a | b | c | a |

We continue by pushing the b, then popping the b, then finally popping the c.

| Step | Description | Input | Stack | Output |
|---|---|---|---|---|
| 0 | Initialize | cab | - | - |
| 1 | Push c | ab | c | - |
| 2 | Push a | b | ac | - |
| 3 | Pop a | b | c | a |
| 4 | Push b | - | bc | a |
| 5 | Pop b | - | c | ab |
| 6 | Pop c | - | - | abc |

We end up with the standard permutation. Through the right combination of pushes and pops, we have sorted the permutation. It turns out that not all of the permutations can be sorted this way; some of them are impossible to sort in one pass through our stack. In this problem your goal is to find the right combination of pushes and pops to sort the input permutation or to report that it is not possible.

## Input

Input will be given with one problem instance per line. Each line will contain an integer $n$ followed by an $n$-character permutation (given as a string). The permutation will contain the first $n$ lower case characters of the alphabet; $1 \leq n \leq 26$. An input of $n = 0$ will signal the end of input.

## Output

For your output, you will print the case number and one of two messages. If the permutation cannot be sorted with one pass through the stack, print "not possible". Otherwise print a string comprised of characters "P" and "p" where a P indicates a push operation (removing a character from the input string and pushing it to the stack), and a p indicates a pop operation (popping a character from the stack and adding it to the output string). The string sequence of P and p characters is unique for sorting a particular input permutation to the standard permutation. Format the output exactly as shown in the example output including the proper whitespace.

## Sample Input

```
3 cab
3 abc
5 dacbe
3 bca
0
```

## Sample Output

```
Case 1: PPpPpp
Case 2: PpPpPp
Case 3: PPpPPpppPp
Case 4: not possible
```

## Problem F: Viva Las Vegas

You have a deck of $n$ playing cards which are numbered $1, 2, \ldots, n$. Assume the deck always starts in sorted order: card 1 is 1, card 2 is 2, ..., card $n$ is $n$.

You will do $m$ *riffle* shuffles. A riffle shuffle splits the deck into two (nearly) equal halves and then puts them into a new deck with interleaving order. More specifically:

1. First split the deck into two halves. If $n$ is even, then both halves contain $\frac{n}{2}$ cards. If $n$ is odd, then the first half contains $\frac{n-1}{2}$ cards and the second half contains $\frac{n+1}{2}$ cards.
2. Now combine the cards into a new deck.
   (a) The first card in the new deck is the first card from the second half.
   (b) The second card in the new deck is the first card from the first half.
   (c) The third card in the new deck is the second card from the second half.
   (d) The fourth card in the new deck is the second card from the first half.
   (e) And so on until all the cards are in the new deck.

For example, if we have an $n = 5$ card deck in original order:

```
1 2 3 4 5
```

we first split it into two halves:

```
1 2       3 4 5
```

and then recombine to obtain:

```
3 1 4 2 5
```

after a riffle shuffle. We now shuffle the deck in this manner $m$ times. After $m$ riffle shuffles, our goal is to find the $k^{th}$ card in the deck.

### Input

Each problem instance will be given on one line of input with three numbers, $n$, $m$, and $k$, where $n$ is the size of the deck, $m$ is the number of riffle shuffles to perform, and $k$ is the location of the card to find at the end. Assume $1 \leq n \leq 100$, $0 \leq m \leq 100$ and $1 \leq k \leq n$ for all input. The end of input will be given by $n = 0$ with no values for $m$ and $k$.

### Output

For each problem, you are to print the case number followed by the $k^{th}$ card after $m$ riffle shuffles of an $n$-card deck, as shown below.

### Sample Input

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

### Sample Output

```
Case 1: 2
Case 2: 3
```