Colombian Collegiate Programming League
CCPL 2019

Round 2 – March 23

Problems

This set contains 11 problems; pages 1 to 18.

(Borrowed from several sources online.)

A - Reaux! Sham! Beaux! 1
B - Game of Stones 3
C - The Punctilious Cruciverbalist 4
D - Roads in the North 6
E - Sequential Yahtzee 7
F - Difference 9
G - Getting Gold 10
H - Roman Holidays 12
I - Cookie 14
J - Two Ends 16
K - Connect the Dots 17

Official site `http://programmingleague.org`

Follow us on Twitter `@CCPL2003`

A - Reaux! Sham! Beaux!

Source file name: `reaux.c`, `reaux.cpp`, `reaux.java`, *or* `reaux.py`

Roshambo – this simple game is known all around the world. In German, it is called “Schnick,Schnack, Schnuck”, in Japanese “Janken”, in Spanish “Cachipún”, in Polish “Papier, kamień, nożyce”.The Czechs call it “Kámen, nůžky, papír”.

Whatever is the name of the game, its principles remain the same. Two players simultaneously form their hand into one of three possible shapes (symbols): Rock (closed fist), Paper (open hand), or Scissors (two fingers extended). If both of them show the same symbol, it is a tie and no points are given. Otherwise, one of the symbols wins: Rock blunts Scissors, Scissors cut Paper, and Paper covers Rock.

Czech Technical University students also know the game very well and use it to resolve small disputes. Imagine, for example, two students living together in one room. Yesterday evening, there was a small celebration, and in the morning, no one wants to go to the lectures. They agreed that one person would be enough to take notices for both, but who will be the poor one? Roshambo is a very effective way to decide.

Did you know there are even the World Series of Roshambo? Our organizing team would like to host the World Championships in 2009. Your task is to help us in developing a Roshambo scoring system and write a program that evaluates one game between two players.

Since the participants will come from different countries, the system must accept input in various languages. The following table shows names of three Roshambo symbols. Note that in some languages,there may be two different words for the same symbol.

Language	**Code**	**Rock**	**Scissors**	**Paper**
Czech	cs	Kamen	Nuzky	Papir
English	en	Rock	Scissors	Paper
French	fr	Pierre	Ciseaux	Feuille
German	de	Stein	Schere	Papier
Hungarian	hu	Ko | Koe	Ollo | Olloo	Papir
Italian	it	Sasso | Roccia	Forbice	Carta | Rete
Japanese	jp	Guu	Choki	Paa
Polish	pl	Kamien	Nozyce	Papier
Spanish	es	Piedr	Tijera	Papel

Input

The input must be read from standard input. The input contains several games. Each game starts with two lines describing players. Each of these two lines contains two lowercase letters specifying the language used by the player (see the language code in the table above), one space, and a player name. The name will consist from at most twenty upper- or lower-case letters.

After the players description, there are at most 100 lines containing individual rounds. Each round is described by two words separated with one space. The words name the symbol shown by the first and second player, respectively. All symbols are named in the mother tongue of the concerned player.All allowed words are shown in the table above, the first letter will be always in uppercase, all other letters in lowercase.

The last round is followed by a line containing one single dash character (‘-’) and then the next game begins. The only exception is the last game in the input, which is terminated by a dot (‘.’)instead of the dash

Output

The output must be written to standard output.

For each game, print five lines of output. The first line should contain the string ‘Game #*G*:’, where *G* is the number of the game, starting with one.

The second line will contain the first player name followed by a colon (‘:’), one space and the number of rounds won by that player. The number should be followed by one space and the word ‘points’. Use the singular form ‘point’ if (and only if) the number of points of the player equals one.

The third line has the same format and shows the second player’s name and points.

The fourth line displays the outcome of the game. It must contain the word ‘`WINNER`’ followed by a colon, space and the name of the player who gained more points. If both players have the same number of points, the fourth line will contain words ‘`TIED GAME`’ instead.

The fifth line is left empty to visually separate individual games.

Note: Acknowledgements to Wikipedia, the free encyclopedia, for providing background information and symbol names in various languages. If your own language is missing, it may be because it has no article on Roshambo in Wikipedia.

Sample Input

```
cs Pepik
en Johnny
Nuzky Scissors
Papir Rock
Papir Scissors
-
de Gertruda
cs Lenka
Stein Papir
Schere Kamen
.
```

Sample Output

```
Game #1:
Pepik: 1 point
Johnny: 1 point
TIED GAME

Game #2:
Gertruda: 0 points
Lenka: 2 points
WINNER: Lenka
```

B - Game of Stones

Source file name: `game.c`, `game.cpp`, `game.java`, *or* `game.py`

Two players, Petyr and Varys, play a game in which players remove stones from N piles in alternating turns. Petyr, in his turn, can remove at most A stones from any pile. Varys, in his turn, can remove at most B stones from any pile. Each player has to remove at least one stone in his turn. The player who removes the last stone wins.

The game has already started and it is Petyr's turn now. Your task is to determine whether he can win the game if both he and Varys play the game in the best possible way.

Input

The input consists of several test cases. The first test case line contains three integers N, A, and B ($1 \leq N \leq 10^5$ and $1 \leq A, B \leq 10^5$). N describes the number of piles and A, B represent Petyr's and Varys' restrictions. The second line contains N integers $X_1, \ldots, X_N$ ($1 \leq X_i \leq 10^6$) specifying the current number of stones in all piles.

The input must be read from standard input.

Output

For each test case, output the name of the winner.

The output must be written to standard output.

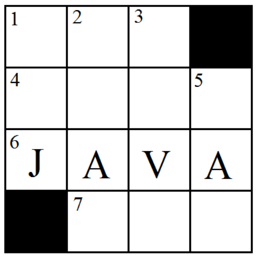
Sample Input	Sample Output

2 3 4
2 3
7 8 9
1 2 3 4 5 6 7
	Petyr
	Varys

C - The Punctilious Cruciverbalist

Source file name: `cruci.c`, `cruci.cpp`, `cruci.java`, *or* `cruci.py`

A crossword puzzle consists of a grid of squares in which intersecting words are placed, one letter per square. Some grid squares are filled black indicating that no letter should go there. Solvers are given clues for each word, and each clue is identified by a starting clue number and either the word Across (for horizontal words) or Down (for vertical words). Each square that can start a word is given a clue number, starting in the upper left and proceeding left-to-right, row-wise. A square will contain an across-clue number if the square to its left is either black or off the grid, and a square will contain a down-clue number if the square above it is either black or off the grid. See the figure below for a simple example.



Will Longz is an avid crossword puzzle enthusiast and as with most solvers, he first tries to solve clues where some of the letters are already present (from already solved clues whose answers intersect the clue he is working on). Typically, the closer to the beginning of the word that these letters are, the better. Will has developed a metric to help him decide the order in which he should solve clues. If the answer to a clue spans n squares, he assigns the value n to the first of these squares, $n - 1$ to the second square, and so on ending with assigning 1 to last square. The rank of any unsolved clue is then defined as follows: for each square that already has a letter in it, add the value assigned to that square to a running total. Then divide this total by the sum of all the values assigned to that clue to get that clue's rank. Once this is done for all the clues, Will solves the clue with the highest rank. If there is a tie, he selects an across clue over a down clue. If there is still a tie he picks the clue with the smallest starting clue number. After each clue is solved Will recalculates the rankings before selecting the next highest ranked clue.

As an example, consider the crossword in the figure above where the 6-Across clue has already been solved (we'll used 6A as a shorter specification for this clue). To determine the rank of the 1-Down (1D) clue, Will first assigns the values 3, 2 and 1 to each of the squares going down. Since the last square has a letter in it, the ranking of this clue is $1/(3 + 2 + 1) = 1/6$. In a similar fashion he determines that the ranking of clues 2D and 3D are $2/10$ and the ranking of 5D is $2/6$. The rankings of the three unsolved across clues 1A, 4A and 7A are all 0. Thus the next clue to solve is 5D (we'll be optimistic and always assume Will can solve any clue). Once this is solved, the rankings of 4A and 7A become $1/10$ and $1/6$, respectively. Since there is now a tie between the highest ranking clues (2D and 3D) and they are both down clues, Will picks the down clue with the lowest clue number and solves 2D. Proceeding similarly, the remaining clues are solved in the following order (with their ranking at time of solution in parentheses): 7A ($4/6$), 4A ($4/10$), 3D ($6/10$) and 1A ($3/6$). Note that clue 1D is not on the list, as it is completely solved by the previously solved clues (namely 1A, 4A and 6A).

For this problem, you will be given a crossword puzzle grid with zero or more squares already filled in and you must determine the order in which unsolved clues should be solved. The filled-in squares may or may not correspond to completely solved clues. One final twist: we will only give you the black squares and letters in the puzzle. You must determine the clue numbers (I bet Will could do it!).

Input

The input consists of several test cases. Each test cases starts with a line containing two integers r c ($1 \leq r, c \leq 50$) specifying the number of rows and columns in the crossword grid. Following this are r lines each containing c characters which describe the puzzle. Each of these characters is either a '`.`' (indicating an empty square), a '`#`' (indicating a black square), or an uppercase alphabetic English character (indicating a solved square). There will always be at least one empty square in the grid.

The input must be read from standard input.

Output

For each test case, display the order in which clues should be solved one per line, using the metric described above. Each clue should start with the clue number followed by either the letter '`A`' or '`D`' (for an across or down clue).

The output must be written to standard output.

Sample Input	**Sample Output**
`4 4`	`5D`
`...#`	`2D`
`....`	`7A`
`JAVA`	`4A`
`#...`	`3D`
	`1A`

D - Roads in the North

Source file name: `roads.c`, `roads.cpp`, `roads.java`, *or* `roads.py`

Building and maintaining roads among communities in the far North is an expensive business. With this in mind, the roads are built in such a way that there is only one route from a village to a village that does not pass through some other village twice.

Given is an area in the far North comprising a number of villages and roads among them such that any village can be reached by road from any other village. Your job is to find the road distance between the two most remote villages in the area.

The area has up to 10,000 villages connected by road segments. The villages are numbered from 1.

Input

The input contains several sets of input. Each set of input is a sequence of lines, each containing three positive integers: the number of a village, the number of a different village, and the length of the road segment connecting the villages in kilometers. All road segments are two-way. Two consecutive sets are separated by a blank line.

The input must be read from standard input.

Output

For each set of input, you are to output a single line containing a single integer: the road distance between the two most remote villages in the area.

The output must be written to standard output.

Sample Input	Sample Output

`5 1 6`
`1 4 5`
`6 3 9`
`2 6 8`
`6 1 7`
	`22`

E - Sequential Yahtzee

Source file name: `yahtzee.c`, `yahtzee.cpp`, `yahtzee.java`, *or* `yahtzee.py`

Danny has a hand-held game of Yahtzee that he likes to play (how 90's!). The object of Yahtzee is to score points by placing the result of rolls of 5 dice into one of 13 categories. The categories are listed below, along with how you score points for each category:

Category	Scoring	Category	Scoring
1's	1 point for each 1	3-of-a-Kind	total of all 5 dice
2's	2 points for each 2	4-of-a-Kind	total of all 5 dice
3's	3 points for each 3	Full House	25
4's	4 points for each 4	Small Straight	30
5's	5 points for each 5	Long Straight	40
6's	6 points for each 6	Chance	total of all 5 dice
		Yahtzee	50

A 3(or 4)-of-a-Kind is any 5 dice where at least three (or four) show the same value. A Full House consists of 3 dice with the same value and the other two with the same value (different from the first value); a Small Straight is four consecutive values on any of four of the dice, a Long Straight is five consecutive values, and a Yahtzee is all five dice showing the same value. Finally the Chance category can be used for any set of five dice. For example, if the five dice showed four 2's and one 5, you would score 8 points if you put it in the 2's category, 5 points if you put it in the 5's category, and 13 points if you put it in either the 3-of-Kind, 4-of-a-Kind or Chance categories. If you put it in any other category you would get 0 points.

A game consists of 13 rounds. At the start of each round, you roll all 5 dice. You can either assign the 5 dice to any of the unused categories or (more likely) re-roll any number of the dice (even all 5 if you like). You get to do this one more time and after (at most) the third roll you must place the dice in an unused category. After 13 rounds, all the categories have been used and you total up the points you got for each category.

In regular Yahtzee you have your choice of which of the scoring categories to use after every round – in fact, the decision on which category to use is one of the challenges of Yahtzee. Danny normally plays this way, but as we said he plays A LOT of Yahtzee, so sometimes he like to switch things up a bit. One of his favorite variations is something he calls *sequential yahtzee*. In this version, the only category you can use after the first set of rolls is the 1's (the first category on his hand-held game); after this, you must use the 2's for your second category, and so on (in the order the categories are given in the table above) until you reach the Yahtzee category.

For example, suppose there's a glitch in Danny's game and the dice only roll 1's (it is a pretty old game). After the first round Danny has (what else) a set of five 1's. In regular Yahtzee he could score 50 points for a Yahtzee, but in sequential yahtzee he must put it in the 1's category and scores 5 points. After he rolls five 1's again, he must put it in the 2's category and scores 0. He scores 0 for the next 4 rounds, putting his five 1's in the 3's, 4's, 5's and 6's category. He gets 5 points for each of the next two rounds, placing his five 1's first in the 3-of-a-Kind and then in the 4-Of-A-Kind. He gets nothing in the next two rounds, scores 5 points for the Chance category and then FINALLY gets 50 points for a Yahtzee in the 13th round. All together he scores 70 points

Danny keeps track of all the dice rolls in the game and often wonders if he could have done better than he did in a game of sequential yahtzee, assuming that the same overall sequence of dice appears regardless of the way he chooses to re-roll dice in any given round. Another example should make things clear. Suppose the sequence of dice is that of the second sample input. If Danny assigns the first five dice to the "1's" category he will get a score of 4, but if he re-rolls the 3, obtaining another 1, his score will improve to 5. In round 2, if he assigns the next 5 dice his score for the "2's" category will be 4, but if he re-rolls the 4, 5, and 6 (obtaining 1, 2, and 3), and then re-rolls these three new dice again, he obtains three more 2s for an improved score of 10 in the second category. At the end of the game, the sequence of five 1s yields a yahtzee; the final 4 is not used. The best score obtainable for this sequence of dice is 340.

Your job is simple: given a sequence of consecutive dice rolls, what's the maximum score possible in a game of sequential yahtzee? Well, maybe it's just the description of the problem that's simple.

Input

The input consists of several test cases. Each test case starts with a line containing an integer n ($65 \leq n \leq 195$) indicating the number of dice rolls. Following this, on one or more lines, are the n dice rolls, all values between 1 and 6 inclusive.

The input must be read from standard input.

Output

For each test case, display the maximum possible sequential yahtzee score using the given dice rolls, filling in all 13 categories. Not all the dice rolls need to be used, but those that are must be consecutive starting with the first roll.

The output must be written to standard output.

Sample Input

```
65
1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1
76
3 1 1 1 1 1 4 2 5 2 6 1 3 5 2 2 2
3 3 3 3 3 4 4 4 4 4 5 5 5 5 5 6 6 6 6 6
6 6 6 6 6 6 6 6 6 1 1 1 2 2 1 2 3 4 5
1 2 3 4 5 1 1 6 1 6 6 6 6 1 1 1 1 1 4
```

Sample Output

```
70
340
```

F - Difference

Source file name: `difference.c`, `difference.cpp`, `difference.java`, *or* `difference.py`

A *smallest different sequence* (SDS) is a sequence of positive integers created as follows: $A_1 = r \geq 1$. For $n > 1$, $A_n = A_{n-1} + d$, where d is the smallest positive integer not yet appearing as a value in the sequence or as a difference between two values already in the sequence. For example, if $A_1 = 1$, then since 2 is the smallest number not in our sequence so far, $A_2 = A_1 + 2 = 3$. Likewise $A_3 = 7$, since $1, 2$ and 3 are already accounted for, either as values in the sequence, or as a difference between two values. Continuing, we have $1, 2, 3, 4, 6$, and 7 accounted for, leaving 5 as our next smallest difference; thus $A_4 = 12$. The next few values in this SDS are $20, 30, 44, 59, 75, 96, \ldots$ For a positive integer m, you are to determine where in the SDS m first appears, either as a value in the SDS or as a difference between two values in the SDS. In the above SDS, $12, 5, 9$ and 11 first appear in step 4.

Input

The input consists of several test cases. Each test case consists of a single line containing two positive integers A_1 m ($1 \leq r \leq 100, 1 \leq m \leq 200\,000\,000$).

The input must be read from standard input.

Output

For each test case, display the smallest value n such that the sequence $A_1, \ldots, A_n$ either contains m as a value in the sequence or as a difference between two values in the sequence. All answers will be $\leq 10\,000$.

The output must be written to standard output.

Sample Input	Sample Output
1 5	4
1 12	4
5 1	2

G - Getting Gold

Source file name: `gold.c`, `gold.cpp`, `gold.java`, *or* `gold.py`

We're building an old-school back-to-basics computer game. It's a very simple text based adventure game where you walk around and try to find treasure, avoiding falling into traps. The game is played on a rectangular grid and the player gets very limited information about her surroundings.

The game will consist of the player moving around on the grid for as long as she likes (or until she falls into a trap). The player can move up, down, left and right (but not diagonally). She will pick up gold if she walks into the same square as the gold is. If the player stands next to (i.e., immediately up, down, left, or right of) one or more traps, she will "sense a draft" but will not know from what direction the draft comes, or how many traps she's near. If she tries to walk into a square containing a wall, she will notice that there is a wall in that direction and remain in the position where she was.

For scoring purposes, we want to show the player how much gold she could have gotten safely. That is, how much gold can a player get playing with an optimal strategy and always being sure that the square she walked into was safe. The player does not have access to the map and the maps are randomly generated for each game so she has no previous knowledge of the game.

Input

The input contains several test cases, each of them as described below. The first line of input contains two positive integers *W* and *H*, neither of them smaller than 3 or larger than 50, giving the width and the height of the map, respectively. The next *H* lines contain *W* characters each, giving the map. The symbols that may occur in a map are as follows:

`P` the player's starting position.

`G` a piece of gold.

`T` a trap.

`#` a wall.

`.` normal floor.

There will be exactly one '`P`'in the map, and the border of the map will always contain walls.

The input must be read from standard input.

Output

For each test case, write to the output the number of pieces of gold the player can get without risking falling into a trap, on a line by itself.

The output must be written to standard output.

Sample Input	**Sample Output**

`7 4`
`#######`
`#P.GTG#`
`#..TGG#`
`#######`
`8 6`
`########`
`#...GTG#`
`#..PG.G#`
`#...G#G#`
`#..TG.G#`
`########`
	`1`
	`4`

H - Roman Holidays

Source file name: `holidays.c`, `holidays.cpp`, `holidays.java`, *or* `holidays.py`

The ancient Romans created many important things: aqueducts, really straight roads, togas, those candles that spout fireworks. But the most useless is Roman numerals, a very awkward way to represent positive integers.

The Roman numeral system uses seven different letters, each representing a different numerical value: the letter I represents the value 1, V 5, X 10, L 50, C 100, D 500 and M 1 000. These can be combined to form the following *base* values:

1	2	3	4	5	6	7	8	9	10
I	II	III	IV	V	VI	VII	VIII	IX	X

10	20	30	40	50	60	70	80	90	100
X	XX	XXX	XL	L	LX	LXX	LXXX	XC	C

100	200	300	400	500	600	700	800	900	1 000
C	CC	CCC	CD	D	DC	DCC	DCCC	CM	M

The Roman numeral representation of a non-base value number x is obtained by first breaking up x into a sum of base values and then translating each base value, largest to smallest. When choosing base values you always choose the largest one $\leq x$ first, then the largest one $\leq$ the amount remaining, and so on. Thus $14 = 10 + 4 =$ XIV, $792 = 700 + 90 + 2 =$ DCCXCII. Numbers larger than 1 000 use as many M's as necessary. So 2 018 = MMXVIII and 1 000 000 would be a string of one thousand M's (hence the word "awkward" in the first paragraph).

The Roman numeral representation gives a new way to order the positive integers. We can now order them alphabetically if we treat the Roman representation of each integer as a word. If one word A is a prefix for another word B then A comes first. We'll call this the *roman ordering* of the positive integers. Thus the first number in roman ordering is `C` (100 in our system). The next three numbers would be `CC`, `CCC` and `CCCI`, and so on.

Note in roman ordering, all numbers larger than 1 000 would come before any number starting with `V` or `X`. Indeed the last number is `XXXVIII`. In this problem you will be given one or more positive integers and must determine their positions in the roman ordering – from the front or back as appropriate.

Input

The input contains several test cases. Each test case starts with a positive integer $n \leq 100$ indicating the number of positive integers to follow, each on a separate line. Each of these remaining numbers will be at most 10^9.

The input must be read from standard input.

Output

For each test case and for each value (other than n), output the position of the integer in the roman ordering, one per line. If the position is relative to the end of the roman ordering, make the integer negative. Thus 38 has roman ordering position -1, 37 has position -2, and so on.

The output must be written to standard output.

Sample Input	Sample Output

```
3
100
101
38
```
	```
	1
	302
	-1
	```

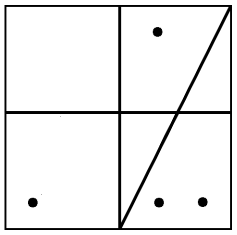
I - Cookie

Source file name: `cookie.c`, `cookie.cpp`, `cookie.java`, *or* `cookie.py`

The *Association of Cookie Makers* (ACM) has developed its newest and most decadent cookie in history, the "Square Chocolate Chip Cookie". This new cookie is a biscuit delight with lots and lots of chocolate chips for all the cookie lovers out there. However, in the development of this revolutionary type of cookie, the chocolate chips sprinklers have had some problems; though all the chips fall on the cookie, they are not evenly spread.

With a tradition of more than 26 years of cookie making, ACM sprinklers cannot be replaced. That is why the sales department has decided to cut the cookie in several pieces to maximize revenue. Each cut is made by a laser that moves across the cookie following a straight line path. Any chip chocolate on the laser path is destroyed during the cut process. Each piece of a square cookie is then valued according to its *chocolate chip density*, defined as the number of chips that it has, divided by its area.

The following figure shows an original cookie with four chips and three cuts. It is apparent that the piece at the lower right corner has a maximal chocolate chip density.



As part of the product testing team at ACM, you have to write a program that detects the *maximal chocolate chip density* among the resulting pieces, given an original square chocolate cookie, the positions on it that received a chocolate chip, and the cuts that should be made.

Input

The input consists of several test cases. The first line in a test case contains three integers L, C, and K, separated by blanks, where L is the length of the cookie side, C is the number of chips, and K is the number of cuts ($2 \leq L \leq 500$, $0 \leq C \leq 5000$, $0 \leq K \leq 50$). Each one of the following C lines contains two blank-separated integers $\hat{x}$ and $\hat{y}$, representing the (x, y) coordinates of a chocolate chip ($0<\hat{x}<L$, $0<\hat{y}<L$) in a Cartesian coordinate system where the cookie is represented by the square with vertices $(0,0)$, $(L,0)$, (L,L), and $(0,L)$. Each one of the following K lines contains four blank-separated integers x_1, y_1, x_2, and y_2, representing two distinct points on a straight line that defines a cut on the same coordinate system ($0 \leq x_1, y_1, x_2, y_2 \leq L$). You may suppose that there are not two chocolate chips placed at the same location. The last test case is followed by a line containing three zeros.

The input must be read from standard input.

Output

For each case, output a single line with a number indicating the maximal chocolate chip density among all the resulting pieces after the cuts. You may suppose that no answer will be greater than 10^2. The answer should be formatted and approximated to three decimal places. The floating point delimiter must be '`.`' (i.e., the dot). The rounding applies towards the *nearest neighbor* unless both neighbors are equidistant, in which case the result is rounded up (e.g., 78.3712 is rounded to 78.371; 78.5766 is rounded to 78.577; 78.3745 is rounded to 78.375, etc.).

The output must be written to standard output.

Sample Input	Sample Output

6 4 3
1 1
4 1
5 1
4 5
3 0 3 6
0 3 6 3
3 0 6 6
0 0 0
	0.296

J - Two Ends

Source file name: `ends.c`, `ends.cpp`, `ends.java`, *or* `ends.py`

In the two-player game “Two Ends”, an even number of cards is laid out in a row. On each card, face up, is written a positive integer. Players take turns removing a card from either end of the row and placing the card in their pile. The player whose cards add up to the highest number wins the game.Now one strategy is to simply pick the card at the end that is the largest — we’ll call this the greedy strategy. However, this is not always optimal, as the following example shows: (The first player would win if she would first pick the 3 instead of the 4.)

3 2 10 4

You are to determine exactly how bad the greedy strategy is for different games when the second player uses it but the first player is free to use any strategy she wishes.

Input

There will be multiple test cases. Each test case will be contained on one line. Each line will start with an even integer n followed by n positive integers. A value of $n = 0$ indicates end of input. You may assume that n is no more than 1000. Furthermore, you may assume that the sum of the numbers in the list does not exceed 10^6.

The input must be read from standard input.

Output

For each test case you should print one line of output of the form:

`In game` m`, the greedy strategy might lose by as many as` p `points.`

where m is the number of the game (starting at game 1) and p is the maximum possible difference between the first player’s score and second player’s score when the second player uses the greedy strategy. When employing the greedy strategy, always take the larger end. If there is a tie, remove the left end.

The output must be written to standard output.

Sample Input	Sample Output

`4 3 2 10 4`
`8 1 2 3 4 5 6 7 8`
`8 2 2 1 5 3 8 7 3`
`0`
	`In game 1, the greedy strategy might lose by as many as 7 points.`
	`In game 2, the greedy strategy might lose by as many as 4 points.`
	`In game 3, the greedy strategy might lose by as many as 5 points.`

K - Connect the Dots

Source file name: connect.c, connect.cpp, connect.java, *or* connect.py

A famous logical problem is that of connecting 9 dots on a paper by drawing 4 line segments with a pencil, while never lifting the pencil from the paper. While this is easy enough (although it requires some thinking outside of the box), Simone has recently been building a game called "Connect the Dots" around a generalisation of the concept.

In Connect the Dots, you are presented with a 4×4 regular grid of dots. Each dot is given a unique number between 1 and 16. The task is then to connect the dots in order by their numbers, starting with dot 1 and ending with dot 16. The dots should be connected using as few line segments as possible, starting at dot 1, with the end of each segment being the start point of the next. The segments are allowed to intersect and overlap one another. Additionally, it is allowed to pass through other points while trying to connect the current point. This means, for example, that visiting the first four points in the order $1, 4, 2, 3, 2, 4, \ldots$ is acceptable. Formally, the sequence $1, 2, \ldots, 15, 16$ must be a subsequence of the sequence of dots visited.

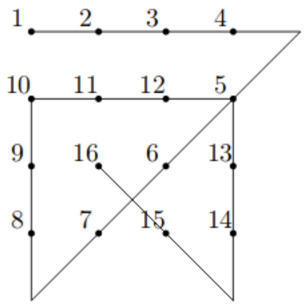


Figure 1: solution to the first sample.

Simone asked you to try the puzzle out, while betting you a balloon that it would be too hard.Prove her wrong by writing a program that solves the puzzle for you!

Input

The input consists of several test cases, each of them as described below.

- 4 lines, each with 4 integers, the numbers of the dots in the grid. The jth number on the ith line is the number of the jth dot in the ith row of the grid of dots.

The 16 numbers in the input are all between 1 and 16 (inclusive) and pairwise distinct.

The input must be read from standard input.

Output

For each test case, output in a single line the minimum number of line segments needed to connect all the dots in order.

The output must be written to standard output.

Sample Input	**Sample Output**
`2`	`6`
`1 2 3 4`	`7`
`10 11 12 5`	
`9 16 6 13`	
`8 7 15 14`	
`1 2 3 4`	
`8 9 10 11`	
`7 15 16 12`	
`6 14 13 5`	