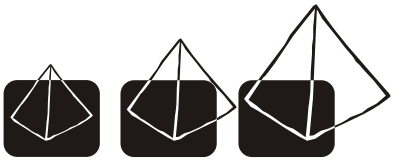
التصفيات الإقليمية الخامسة للدول العربية وشمال أفريقيا

الجامعة الأمريكية بالقاهرة، ٢٠٠٢



Arab and North African Region
American University in Cairo
October 31 - November 3, 2002

The 27th Annual ACM International Collegiate Programming Contest

Arab and North African 5th Regional Contest

sponsored by IBM, Microsoft

American University in Cairo November 2, 2002

The problem set is made of 13 numbered pages

The maximum allowable running time for a submission is 120 seconds.

[A] Anagrams

Program:	`anagrams.(c|cpp|java)`
Input:	`anagrams.in`
Output:	`anagrams.out`

Description

Two words are said to be *anagrams* if one can be formed by permuting the letters of the other. For example: `"pots"`, `"tops"`, and `"stop"` are anagrams. An *anagram chain* is a list of words that are all anagrams to each other. The shortest anagram chain has the length two. We're interested in calculating the length of the longest anagram chain in a given list of words. For example, the following nine words: `rates`, `pots`, `tops`, `along`, `aster`, `stop`, `stare`, `tears`, and `long` has two anagram chains where the longest includes the four words: `rates`, `aster`, `stare`, and `tears`.

Input Format

Your program will be tested on a number of test cases. The first line of the input file contains an integer `D` representing the number of test cases in the input file.

Each test case contains one or more words, but no more than 20,000 words, with no duplicates. Each word appears on a separate line. All words are in small letters, and in no particular order. No word will be longer than 10 characters. Each test case ends with a string made of one or more `'-'` characters.

Output Format

For each test case, write, on a separate line, the length (number of words) of the longest anagram chain found in the given list of words.

Sample Input/Output

anagrams.in

```
2
rates
pots
tops
along
aster
stop
stare
tears
long
-----
north
fresher
refresh
thorn
bye
--
```

anagrams.out

```
4
2
```

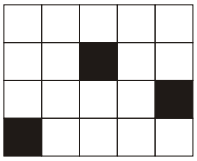
[B] Crossing the Words

Program:	crossword.(c|cpp|java)
Input:	crossword.in
Output:	crossword.out

Description

Yasmeen loves solving crossword puzzles. A crossword puzzle is a puzzle in which words are filled into a pattern of numbered squares (the grid) in answer to correspondingly numbered clues and in such a way that the words read across (left to right,) and down.
Yasmeen normally solves a puzzle by writing the answers on a separate sheet of paper rather than on the puzzle itself. Once the puzzle is solved, she then neatly copies the solution on the grid.
But today Yasmeen has a problem, she lost the page of the clues (which describes where the words go on the puzzle.) All what she has is the puzzle grid and the list of words that constitutes the answers to the clues. She needs your help in reconstructing the solution to the puzzle.

he stood no
ones else so
she tell do lost



s	t	o	o	d
h	e	■	n	o
e	l	s	e	■
■	l	o	s	t

Input Format

The first line of the input file is an integer D representing the number of test cases in the input file.

The first line of each test case specifies three integers: R, C, and W. R is the number of rows in the grid, C is the number of columns in the grid, and W is the number of words. Note that $0 < R < 50$ and $0 < C < 50$ and $0 < W < 1000$.

Starting at the second line of each test case are R lines describing the grid. The grid is described using two characters: A '.' indicates a square that should be eventually filled with a letter, while a '#' means the corresponding square "is blocked" (it doesn't receive any letter.)

The list of words appears on the last line of each test case. Words are separated by exactly one space character. All words are small letters and there are no duplicates.

The sample input describes two test cases: The first is a 4x5 grid with 10 words. The second test case starts at line #8 and describes a 7x7 grid with 15 words.

Output Format

For each test case, print the grid with the solution filled in. There should be a blank line after each grid.

Sample Input

crossword.in

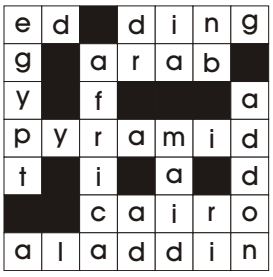
```
2
4 5 10
.....
..#..
....#
#....
he stood no ones else so she tell do lost
7 7 15
..#....
.#....#
.#.###.
.......
.#.#.#.
##.....
.......
egypt arab africa pyramid cairo dr ad nb ri ia ed maid addon ding aladdin
```

Sample Output

crossword.out

```
stood
he#no
else#
#lost

ed#ding
g#arab#
y#f###a
pyramid
t#i#a#d
##cairo
aladdin
```



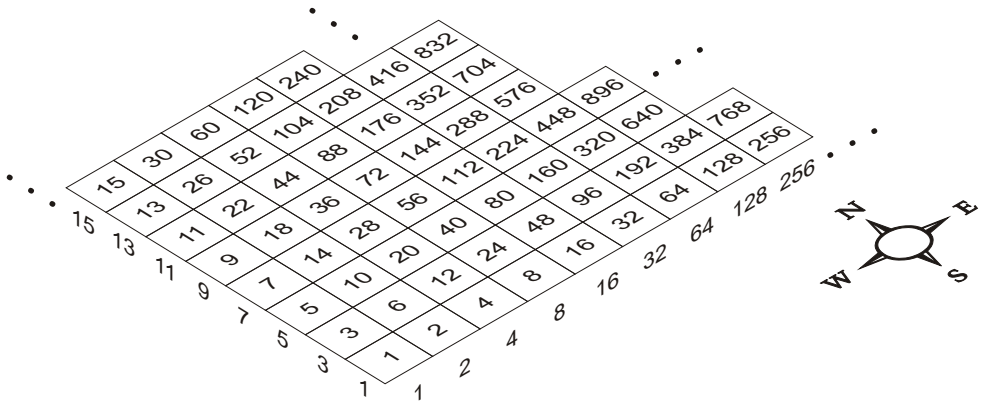
Sample CrossWord #2

[C] City of Flatland

Program:	`flatland.(c|cpp|java)`
Input:	`flatland.in`
Output:	`flatland.out`

Description

In recognition to the number of famous mathematicians of its residents, the City of Flatland has decided to rename all its streets as numbers (positive integers to be more precise.) The streets of Flatland are organized as a grid. The city decided to number all its North-South streets using powers of two $(1, 2, 4, 8, \ldots)$ and all its East-West streets using odd numbers $(1, 3, 5, \ldots)$. The city also decided to re-number all its buildings so that the number of each building is the result of multiplying the numbers of the two streets the building is on. For example, building #40 is at the intersection of streets 5 and 8.



The problem with this numbering scheme is that it is not easy for the residents to determine the distance between buildings. The distance between any two buildings is the number of buildings one needs to cross to go from one building to another. One can only move parallel to the streets (no diagonals or any other shortcuts.) For example, to go from building #6 to building #40, one has to travel one building north and two buildings east, so the distance is 3. Similarly, the distance from building #80 to building #88 is 4.

Help the residents of Flatland by writing a program that calculates the distance between any two given buildings.

Input Format

The input is made of one or more pairs of building numbers. Each pair `<S,T>` appears on a single line with a single space between the two numbers. Note that `S,T < 1,000,000,000`. The end of the input is identified by the pair `<0,0>` (which is not part of the test cases.)

Output Format

For each input pair `<S,T>`, the output file should include a line of the form:

`The distance between S and T is D.`

The output file should be in the same order as the input file.

Sample Input/Output

flatland.in

```
12 14
20 30
40 50
0 0
```

flatland.out

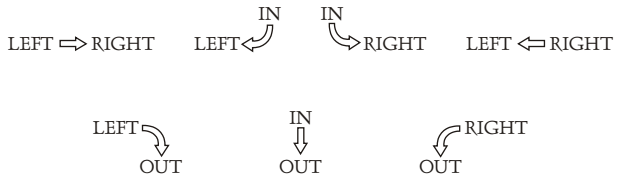
```
The distance between 12 and 14 is 3.
The distance between 20 and 30 is 6.
The distance between 40 and 50 is 12.
```

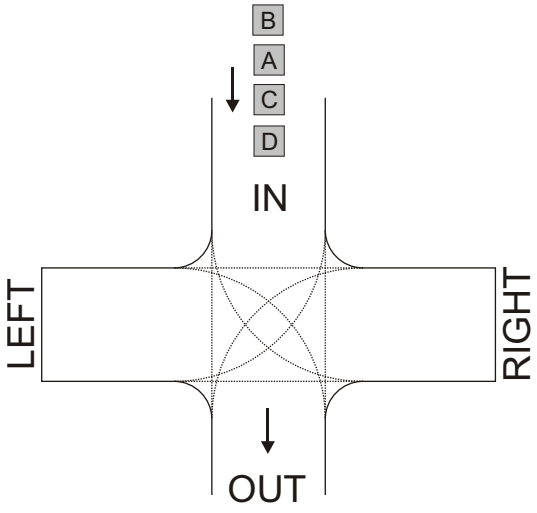
[D] Pushing Carts

Program:	`carts.(c|cpp|java)`
Input:	`carts.in`
Output:	`carts.out`

Description

Lamees works at the delivery zone in a plant. Carts arrive to the delivery zone on the [IN] track, and it is Lamees's responsibility to make sure the carts are sorted before leaving the delivery zone. The carts are heavy, so they must remain on the tracks at all times. The carts can be temporarily pushed to either of the two side bays. The exchange at the center can be configured to move the carts in the following seven directions:





Initially, the exchange is set to [IN → OUT]. The exchange can be reconfigured from the control room which is a bit far from the delivery zone. Normally, there is an operator at the control room to configure the exchange while Lamees remains at the delivery zone. But today the operator has called in sick, and Lamees has to do both tasks herself. Every time the exchange is to be reconfigured, Lamees has to make a trip to the control room. Help Lamees by writing a program to compute the minimum number of times the exchange must be reconfigured to complete the job.

For example, the carts `D-C-A-B` can be sorted in three exchanges (see figure on next page.) First set the exchange to [IN → LEFT], push `D` and `C`, then set the exchange to [IN → OUT], push `A,B`, finally set the exchange to [LEFT → OUT] and push `C,D`.

Input Format

The input file includes a number of test cases. Each test case is described using a string of capital letters appearing on a separate line. The end of the test cases is indicated by a string starting with the letter 'Z' (which is not part of the test cases.)

Each test case specifies the order in which the carts arrive to the [IN] track. For example, the string `"DCAB"` says that `D` arrives first, then `C`, then `A`, and finally `B`. The proper order of the carts is a lexicographic ordering (i.e. `"ABCD"`). There will be no "missing" letters in any test case, and no duplicates. There will be no more than 26 carts in any test string.

Output Format

For each test case, write on a separate line, the minimum number of exchanges needed to sort the carts.

Sample Input/Output

carts.in

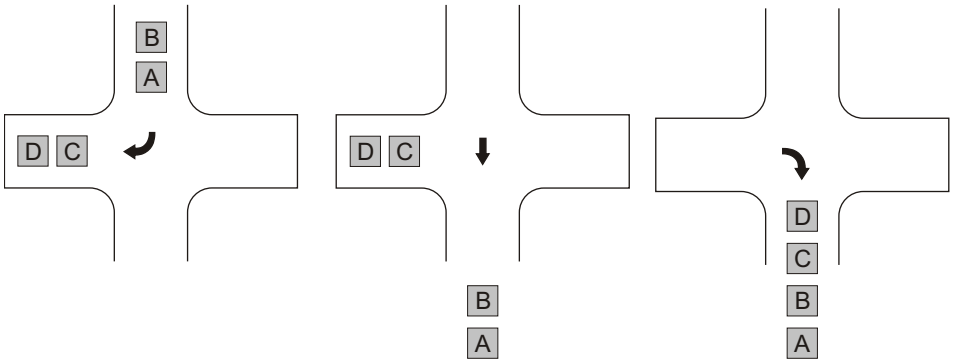
```
DCAB
EDACB
ZaEnd
```

carts.out

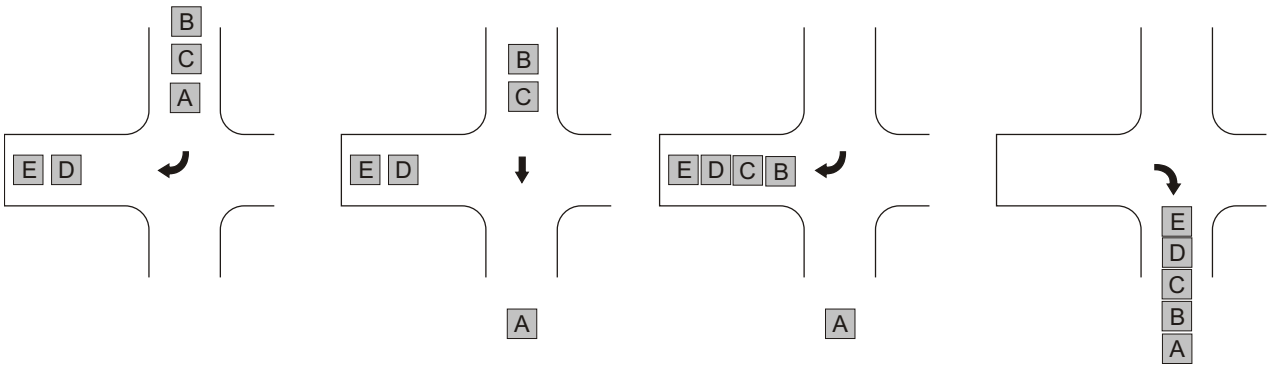
```
3
4
```

Illustrations

How the first test case was solved in 3 moves:



How the second test case was solved in 4 moves:



[E] Palindromes

Program:	palindromes.(c|cpp|java)
Input:	palindromes.in
Output:	palindromes.out

Description

A palindrome is a word that reads the same forward or backward. For example, "noon", "civic", and "rotator" are all palindromes. We can extend the definition of palindromes to include integers too. For example, "5", "22", and "10701" are all palindromes. As for negative integers, we'll say that a negative integer is a palindrome only if its positive counterpart is a palindrome.

Write a program that computes how many palindrome integers there are between any two given integers.

Input Format

The input file is made of a number of test cases. Each test case specifies a range of integers using a pair of integers `<L,U>` where $-1,000,000 < L \leq U < 1,000,000$. Each test case is specified on a separate line, with at least one space character between `L` and `U`.

The set of test cases ends with the pair `<-1,-1>`, which is not part of the test cases.

Output Format

For each test case `<L,U>`, your program should print how many palindromes there are within the range `L` and `U` (inclusive).

Sample Input/Output

palindromes.in

```
101 202
11 30
-202 -101
-1 -1
```

palindromes.out

```
11
2
11
```

[F] Why Johnny Can't Count

Program:	`johnny.(c|cpp|java)`
Input:	`johnny.in`
Output:	`johnny.out`

Description

Poor Johnny; He can hardly count. Johnny needs a program to "spell out" numbers into their equivalent English text. For example, the number 109210 is read in English as: *"one hundred and nine thousand, two hundred and ten"*. To make the program easier, Johnny is willing to accept the following compromises:

1. The program will be given positive integers less than a million.
2. No need to print any punctuation marks.
3. Use singular words, not plural. For example: *"thousand"* rather than *"thousands"*.
4. Don't use the word *"and"* in the phrase. For example, instead of converting the number 102 into *"one hundred and two"*, all you need to do is convert it to *"one hundred two"*. Similarly, 109210 would be spelled out as: *"one hundred nine thousand two hundred ten"*.

On the other hand, Johnny requires the following:

1. Johnny hates spelling mistakes. The output should be spelled correctly and using only small letters. The list of allowable words in the output is:
   `zero, one, two, three, four, five, six, seven, eight, nine, ten, eleven, twelve, thirteen, fourteen, fifteen, sixteen, seventeen, eighteen, nineteen, twenty, thirty, forty, fifty, sixty, seventy, eighty, ninety, hundred, thousand.`
2. Johnny requires that the output be sorted in a non-decreasing order to make it easier for him to locate the numbers.
3. For an unclear reason, there will be duplicates in the input file, and Johnny insists that each instance be printed.

Input Format

The input is made of one or more positive integers terminated by a negative number (which is not part of the input data.) Each number appears on a separate line.

Output Format

For each number, you should print exactly one line showing the number and its equivalent phrase as shown in the sample output. Use a single space character as a separator. Note the colon character `':'` after the number.

Sample Input

johnny.in

```
199
123456
14
199
421000
112
999999
199
-1
```

Sample Output

johnny.out

```
14: fourteen
112: one hundred twelve
199: one hundred ninety nine
199: one hundred ninety nine
199: one hundred ninety nine
123456: one hundred twenty three thousand four hundred fifty six
421000: four hundred twenty one thousand
999999: nine hundred ninety nine thousand nine hundred ninety nine
```

[G] Snakes & Ladders

Program:	`snakes.(c|cpp|java)`
Input:	`snakes.in`
Output:	`snakes.out`

Description

Snakes and ladders is a board game played on an $N * N$ squares grid (see the figure on the next page.) The squares are numbered from 1 up to N^2. Players start by placing their counters at square #1. Players take turns by throwing a dice and moving their counter the number of spaces shown on the dice. The board includes a number of snakes and ladders. If a player's counter lands on the mouth of a snake, the counter must move down to the tail of the snake. If a player's counter lands at the bottom of a ladder then the counter must climb to the top. The winner is the first player to reach square $\#N^2$. The following points are worth knowing about the layout of the board:

1. There are no ladders or snakes that start or end at the first or last square.
2. Snakes and ladders can't be adjacent. There is at least one "regular" square between any two squares that are the starting or ending points of either snakes or ladders.

Your friend Fadi wants you to write a program to help him win the game of Snake and Ladders. See, Fadi is a professional cheater. He can throw the dice and let it show any number he desires. Fadi wants a program to determine the minimum number of throws needed to win the game.

For example, given the "example one" board shown on the next page, Fadi can win in three moves as follows: On the first throw he gets a 4, moving him to square 5, up the ladder to square 16. Then another 4 on the second throw taking him to square 20, up the ladder to square 33. A 3 on the third throw wins him the game.

Input Format

The first line of the input file is an integer `D` representing the number of test cases in the input file.

Each test case is described using three lines. The first line includes three integers: `N`, `S`, and `L`. `N` is the size of the board, `S` is the number of snakes on the board, `L` is the number of ladders. Note that $0 < N \leq 20$ and $0 < S < 100$ and $0 < T < 100$.

The second line of a test case includes `S` integer pairs. Each pair describes a particular snake. The first integer is the starting square of the snake (its mouth) and the second integer is the ending square (the tail.) Remember, squares are numbered starting at 1.

The third line is the same as the second but for ladders. It includes `L` integer pairs.

Output Format

For each test case, write, on a separate line, the minimum number of dice throws required to win the game.

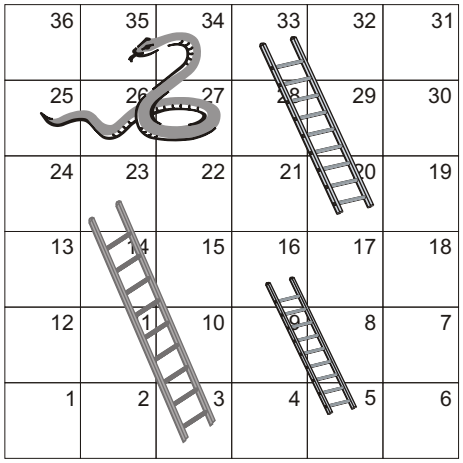
Sample Input/Output

snakes.in

```
2
6 1 3
35 25
3 23 5 16 20 33
5 1 1
16 14
9 11
```

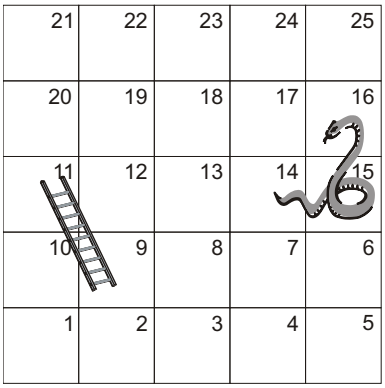
snakes.out

```
3
4
```



$N = 6$

Example One



$N = 5$

Example Two

[H] Prime Spiral

Program:	spiral.(c|cpp|java)
Input:	spiral.in
Output:	spiral.out

Description

Back in 1963, while doodling during a boring talk at a scientific meeting, a Polish-American mathematician named Stanislaw Ulman came up with what is now known as the *Prime Spiral*. While drawing a grid of lines, he decided to number the intersections according to a spiral pattern as you see in the adjacent figure. He then began circling the numbers in the spiral that were primes. Surprisingly, the circled primes appeared to fall along a number of diagonal straight lines.

17	16	15	14	13
18	5	4	3	12
19	6	1	2	11
20	7	8	9	10
21	22	23	24	25

Prime Spiral of Size N=5

In this problem, we're interested in finding the largest sum of primes along any diagonal straight line for any given grid of size $N < 100$. For example, inspecting the prime spiral for $N = 5$, the largest sum of primes on a diagonal line is $19 + 7 + 23 = 49$. Similarly the largest sum of primes for $N = 3$ is 10.

5	4	3
6	1	2
7	8	9

N=3

Input Format

The first line in the input file contains a single integer D which represents how many data sets are used to test your program.

Each data set contains exactly one integer, N, representing the size of the grid, on a separate line. N is always an odd number in the range $0 < N < 100$

Output Format

For each test case, write the largest sum on a separate line.

Sample Input/Output

spiral.in

```
2
5
3
```

spiral.out

```
49
10
```