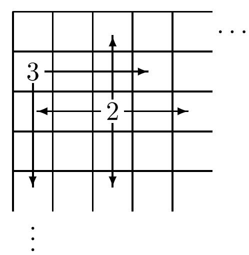
Spring 1992 Denison Computer Science Contest
Problem 1
A-maze-in' Numbers

The object of this problem is to find the shortest number of moves to navigate through a maze represented by an 8 by 8 grid of numbers. The start location is the upper lefthand corner square $(1,1)$ and the finish location is the lower righthand corner square $(8,8)$. The value in each square indicates how far you can move from that square in either of four directions: up, down, left or right. Moves off the board are illegal. For example, from square $(3,3)$ below you can move to squares $(1,3),(3,1),(5,3)$ and $(3,5)$ (corresponding to directions up, left, down and right), while from square $(2,1)$ you can only move to squares $(5,1)$ and $(2,4)$ (down and right).



Input for this problem will consist of 8 rows of numbers, each row consisting of 8 integers separated by a blankspace (the first row corresponds to row 1 in the maze, the second to row 2, etc). Each integer will lie in the range `[0..7]`. Output should consist of 1 line which lists all the squares in the shortest path in the order visited (including the start and finish squares). Each square should be of the format `(row#,col#)`. Do not use any space or commas to separate each square. If no path exists between the start and finish locations, you should print the message `No path through the maze exists`.

Example

The following input

```
5 2 1 3 5 4 2 0
1 0 4 1 1 6 2 6
2 6 3 2 6 3 4 3
7 1 5 2 7 4 1 7
2 6 3 7 4 3 2 6
0 7 3 0 1 2 3 1
7 4 1 5 7 5 5 6
2 1 5 2 4 7 3 5
```

should produce the following output:

```
(1,1)(1,6)(5,6)(5,3)(8,3)(8,8)
```

Spring 1992 Denison Computer Science Contest
Problem 2
Piazza Resistance

Many European cities are designed with the older section of the city, usually located close to the center of town, separated by "walls" (either physically or conceptually) from the newer sections of town. In this problem, you will be given a description of such a city and must list all points that fall within the older part of town. The town will be represented by a collection of major roads and the points of interest are the intersections of these roads; we'll call the intersections *piazzas* to give them an European flavor. Besides the roads, you will be given a list of those piazzas that represent the wall. You will also be given one piazza P called the *city center* located in the older part of the town. Given this information, you must find all other piazzas in the older part town. A piazza is in the older part of town if there exists at least one path between it and P that does not use any of the wall piazzas. Note that no wall piazza is consider to be in the older part of town.

The first line of input will contain two numbers: n = the number of piazzas and m = the number of roads connecting the piazzas. For all problems, $n \leq 20, m \leq 100$. The piazza's are assumed to be numbered from 1 to n. The next m lines will contain the roads, one per line, specified by the two piazzas they connect. For example, the line `5 14` would indicate that there exists a road between piazza 5 and piazza 14. All roads are considered two-way. After the roads specifications will be two final lines of input. The first will list the piazzas that make up the wall, and the last line will contain the city center piazza P. Your output should consist of two lines: the first should contain the phrase `List of all piazzas in the older part of town`, and the second line should list these piazzas (including the initial inside piazza given) in ascending order, separated by commas.

Example

The following input (the comments are added for clarity and will NOT be in the actual input)

```
8 9                     (* number of piazzas and number of roads *)
1 3                     (* road list *)
3 7
5 1
5 2
4 6
6 7
7 5
5 4
6 8
1 3 5 7                 (* wall *)
4                       (* city center *)
```

should produce the following output:

```
List of all piazzas in the older part of town
4, 6, 8
```

Spring 1992 Denison Computer Science Contest
Problem 3
Prime Time

We define an *integer substring* of a positive integer to be any integer formed by consecutive digits in the original integer. For example, the integer 6158 contains the integer substrings 6, 1, 5, 8, 61, 15, 58, 615, 158 and 6158. The object of this program is to find the longest prime integer substring for any given integer. In case of ties we will choose the largest of the longest prime integer substrings.

For this problem each line of input will contain one positive integer followed immediately by an end-of-line. Each integer will be ≤ 2147483647 (this value is the maximum value that can be stored in variables of type `longint`; don't try to store the input in a variable of type `integer`). For each input, you should output one line in the following format:

```
<input_integer>: <longest_prime_substring>
```

where <longest_prime_substring> is equal to the longest prime substring found in <input_integer>, or the message `No prime integer substring` if none exists. Your output of the longest prime substring must include any leading `0`'s if they are present in the input integer (see example below). End-of-file will terminate input.

Example

The following input

```
2319
16673
287
987654321
35001
6804
```

should produce the following output:

```
2319: 31
16673: 16673
287: 7
987654321: 76543
35001: 001
6804: No prime integer substring
```

Spring 1992 Denison Computer Science Contest
Problem 4
Countdown

In this problem you are to count the occurrences of the 26 letters of the alphabet in a given text and output the results in a format described below. For this problem we will treat the upper and lowercase versions of each letter as the same letter. All other characters in the text are to be ignored.

Input for this problem will be several lines of text terminated by end-of-file. After input has terminated, the letter distribution should be printed in the following format: The first row consists of 26 upper case letters separated by one space. Vertically beneath each letter is the number of occurrences of the letter (you may assume each letter occurs less than 1000 times).

Example

The following input:

```
A skunk sat on a stump.
The stump thunk the skunk stunk.
The skunk thunk the stump stunk.
```

should produce the following output:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
3 0 0 0 4 0 0 6 0 0 1 0 3 8 1 3 0 0 9 1 1 0 0 0 0 0
                    0                 2 0
```

Spring 1992 Denison Computer Science Contest
Problem 5
Buzz vs. the Robots

Spaceman Buzz Boffo is stranded on the planet Gridlock, a flat, square planet divided (conveniently) into a 40 by 40 grid. Gridlock is inhabited by deadly robots that each second march relentlessly towards any invader – Buzz, in this case. Each second Buzz has 5 options – move north, south, east, west or stay put. The robots will then move (if possible) 1 square closer to Buzz in the east-west direction and (if possible) 1 square closer in the north-south direction. All moves are made simultaneously and will never cause a robot or Buzz to move off the surface of Gridlock.

Fortunately for Buzz, if two or more robots try to occupy the same square they crash, leaving a scrap heap. Any robots moving onto a scrap heap will also be destroyed. Buzz may not move onto a scrap heap; if he attempts to, he will instead remain on the square he currently occupies. If one or more robots occupy the same square as Buzz, he's history. Buzz hopes to survive long enough to be rescued.

The first line of input is an integer n (≤ 20) indicating the number of robots on Gridlock. The next n lines each contain two integers (both ≥ 1 and ≤ 40) indicating the coordinates of each robot. The next line will contain Buzz's starting coordinates. (The far north-west corner of Gridlock has coordinates $(1, 1)$.) You may assume that the n robots and Buzz initially occupy $n + 1$ distinct squares. The final line of input is a nonempty string of characters from the set {`N,S,E,W,.`}. (This line terminates on end-of-line.) These indicate Buzz's series of moves. (`N`=north, `S`=south, `E`=east, `W`=west and `.`=stay put.) The moves given are guaranteed not to move Buzz off Gridlock. The output should consist of either an integer indicating how many moves Buzz made before he met his fate, or the message `BUZZ SURVIVES!` if Buzz is able to make all his moves from the input and avoid the robots.

We give a small example below with only an 8 by 8 grid. In this example `B` = Buzz, `@` = a robot, `*` = a scrapheap and `.` = unoccupied space. Suppose Buzz's list of moves is `WN.SE`:

```
  ........       ........      ........
  .@......       ........      ...B@...
  ....B...       ..*B.@..      ..*.....
  .@....@.       .@......      ...*....
  @.......       ..@@@...      ...@....
  .@.@.@..       ...@....      ........
  ....@...       ........      ........
  ........       ........      ........
initial pos    after W move  after N move
```

Buzz is then killed on the third step by the robot to his right, so the output in this case would be `3`. Note that in this example there were more moves provided for Buzz than are actually executed.

Example

The following input

```
8
2 2
4 2
4 7
5 1
6 2
6 4
6 6
7 5
3 5
WN.SE
```

should produce the following output:

```
3
```

If the robot initially at `4 7` started at `8 7` instead the output would be `BUZZ SURVIVES!`.