# 2001 Mid-Atlantic Regional Programming Contest

Welcome to the 2001 Programming Contest. Before you start the contest, please be aware of the following notes:

1. There are eight (8) problems in the packet, using letters A-H. These problems are NOT necessarily sorted by difficulty. As a team's solution is judged correct, the team will be awarded a balloon. The balloon colors are as follows:

| Problem | Problem Name | Balloon Color |
|---|---|---|
| A | Financial Management | Red |
| B | I Think I Need a Houseboat | Pink |
| C | Start Up The Startup | Yellow |
| D | A New Growth Industry | Green |
| E | Algernon's Noxious Emissions | Blue |
| F | FDNY to the Rescue | Purple |
| G | For the Porsche | Silver |
| H | Oh, Those Achin' Feet | Gold |

2. **All solutions must read from standard input and write to standard output.** In C this is scanf/printf, in C++ this is cin/cout, and in Java this is System.in/System.out. From your workstation you may test your program with an input file by redirecting input from a file:
```
program < file.in
```
3. Solutions for problems submitted for judging are called runs. Each run will be judged. Runs for each particular problem will be judged in the order they are received. However, it is possible that runs for different problems may be judged out of order. For example, you may submit a run for B followed by a run for C, but receive the response for C first. DO NOT request clarifications on when a response will be returned. If you have not received a response for a run within 90 minutes of submitting it, you may ask the site judge to determine the cause of the delay.

   The judges will respond with one of the following responses. In the event that more than one response is applicable, the judges may respond with any of the applicable responses.

| Response | Description |
|---|---|
| Correct | The run has been judged correct. |
| Incorrect Output | The program generated output that is not correct. |
| Incorrect Output Format | The program's output is not in the correct format |
| Incomplete Output | The program failed to generate all required output |
| Syntax Error | The program failed to compile on the judges machine |
| Run-Time Error | The program experienced a run time error. |
| Time-Limit Exceeded | The program failed to complete within two minutes. |

4. In the event that you feel a problem statement is ambiguous, you may request a clarification. **Read the problem carefully before requesting a clarification.** If the judges do not believe that you have discovered an ambiguity in the problem, you will receive the response, “The problem statement is not ambiguous, no clarification is necessary.” If you receive this response, you should read the problem description more carefully. If you still feel there is an ambiguity, you will have to be more specific or descriptive of the ambiguity you have found. If the problem statement is ambiguous in specifying the correct output for particular input, please include that input data in the clarification request.
5. **The submission of abusive programs or clarification requests to the judges will be considered grounds for immediate disqualification.**
6. Good luck, and HAVE FUN!!!

# Problem A: Financial Management

Larry graduated this year and finally has a job. He's making a lot of money, but somehow never seems to have enough. Larry has decided that he needs to grab hold of his financial portfolio and solve his financing problems. The first step is to figure out what's been going on with his money. Larry has his bank account statements and wants to see how much money he has. Help Larry by writing a program to take his closing balance from each of the past twelve months and calculate his average account balance.

## Input Format:

The input will be twelve lines. Each line will contain the closing balance of his bank account for a particular month. Each number will be positive and displayed to the penny. No dollar sign will be included.

## Output Format:

The output will be a single number, the average (mean) of the closing balances for the twelve months. It will be rounded to the nearest penny, preceded immediately by a dollar sign, and followed by the end-of-line. There will be no other spaces or characters in the output.

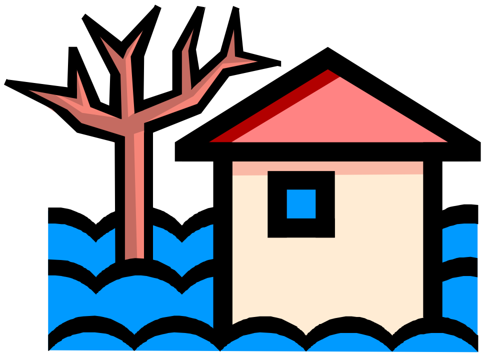
## Sample Input:

```
100.00
489.12
12454.12
1234.10
823.05
109.20
5.27
1542.25
839.18
83.99
1295.01
1.75
```

## Sample Output:

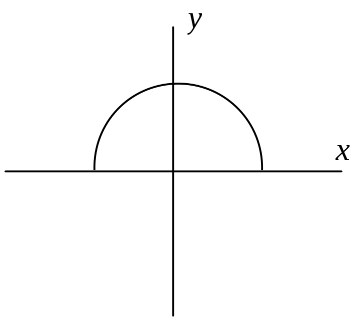
```
$1581.42
```

# Problem B: I Think I Need a Houseboat



Fred Mapper is considering purchasing some land in Louisiana to build his house on. In the process of investigating the land, he learned that the state of Louisiana is actually shrinking by 50 square miles each year, due to erosion caused by the Mississippi River. Since Fred is hoping to live in this house the rest of his life, he needs to know if his land is going to be lost to erosion.

After doing more research, Fred has learned that the land that is being lost forms a semicircle. This semicircle is part of a circle centered at (0,0), with the line that bisects the circle being the X axis. Locations below the X axis are in the water. The semicircle has an area of 0 at the beginning of year 1. (Semicircle illustrated in the Figure.)



## Input Format:

The first line of input will be a positive integer indicating how many data sets will be included (N).
Each of the next N lines will contain the X and Y Cartesian coordinates of the land Fred is considering. These will be floating point numbers measured in miles. The Y coordinate will be non-negative. (0,0) will not be given.

## Output Format:

For each data set, a single line of output should appear. This line should take the form of:
"Property N: This property will begin eroding in year Z."
Where N is the data set (counting from 1), and Z is the first year (start from 1) this property will be within the semicircle AT THE END OF YEAR Z. Z must be an integer.
After the last data set, this should print out "END OF OUTPUT."

## Notes:

1. No property will appear exactly on the semicircle boundary: it will either be inside or outside.
2. This problem will be judged automatically. Your answer must match exactly, including the capitalization, punctuation, and white-space. This includes the periods at the ends of the lines.
3. All locations are given in miles.

## Sample Input:

```
2
1.0 1.0
25.0 0.0
```

## Sample Output:

```
Property 1: This property will begin eroding in year 1.
Property 2: This property will begin eroding in year 20.
END OF OUTPUT.
```

## Problem C: Start Up the Startup

Clearly the economy is bound to pick up again soon. As a forward-thinking Internet entrepreneur, you think that the 'Net will need a new search engine to serve all the people buying new computers. Because you're frustrated with the poor results most search engines produce, your search engine will be better.

You've come up with what you believe is an innovative approach to document matching. By giving weight to the number of times a term appears in both the search string and in the document being checked, you believe you can produce a more accurate search result.

Your program will be given a search string, followed by a set of documents. You will calculate the score for each document and print it to output in the order the document appears in the input file. To calculate the score for a document you must first calculate the term score for each term appearing in the search string. A term score is the number of times a term occurs in the search string multiplied by the number of times it occurs in the document. The document score is the sum of the square roots of each term score.

### Input Format:

The input file consists of a set of documents separated by single lines containing only ten dashes, "----------". No line will be longer than 250 characters. No document will be longer than 100 lines. The first document is the search string. The input file terminates with two lines of ten dashes in a row.

The input documents will use the full ASCII character set. You must parse each document into a set of terms.

Terms are separated by whitespace in the input document. Comparisons between terms are case-insensitive. Punctuation is removed from terms prior to comparisons, e.g. "don't" becomes "dont". The resulting terms should contain only the characters {[a-z],[0-9]}. A term in the input consisting only of punctuation should be ignored. You may assume the search string and each document will have at least one valid term.

### Output Format:

The output is a series of scores, one per line, printed to two decimal places. The scores are printed in the order the documents occur in the input. No other characters may appear in the output.

## Sample Input:

```
fee fi fo fum
----------
fee, fi, fo! fum!!
----------
fee fee fi, me me me
----------
----------
```

## Sample Output:

```
4.00
2.41
```

## Problem D: A New Growth Industry

A biologist experimenting with DNA modification of bacteria has found a way to make bacterial colonies sensitive to the surrounding population density. By changing the DNA, he is able to "program" the bacteria to respond to the varying densities in their immediate neighborhood.

The culture dish is a square, divided into 400 smaller squares (20x20). Population in each small square is measured on a four point scale (from 0 to 3). The DNA information is represented as an array D, indexed from 0 to 15, of integer values and is interpreted as follows:

In any given culture dish square, let K be the sum of that square's density and the densities of the four squares immediately to the left, right, above and below that square (squares outside the dish are considered to have density 0). Then, by the next day, that dish square's density will change by D[K] (which may be a positive, negative, or zero value). The total density cannot, however, exceed 3 nor drop below 0.

Now, clearly, some DNA programs cause all the bacteria to die off (e.g., [-3, -3, …, -3]). Others result in immediate population explosions (e.g., [3,3,3, …, 3]), and others are just plain boring (e.g., [0, 0, … 0]). The biologist is interested in how some of the less obvious DNA programs might behave.

Write a program to simulate the culture growth, reading in the number of days to be simulated, the DNA rules, and the initial population densities of the dish.

## Input Format:

Input to this program consists of three parts:

1. The first line will contain a single integer denoting the number of days to be simulated.
2. The second line will contain the DNA rule D as 16 integer values, ordered from D[0] to D[15], separated from one another by one or more blanks. Each integer will be in the range -3…3, inclusive.
3. The remaining twenty lines of input will describe the initial population density in the culture dish. Each line describes one row of squares in the culture dish, and will contain 20 integers in the range 0…3, separated from one another by 1 or more blanks.

## Output Format:

The program will produce exactly 20 lines of output, describing the population densities in the culture dish at the end of the simulation. Each line represents a row of squares in the culture dish, and will consist of 20 characters, plus the usual end-of-line terminator.

Each character will represent the population density at a single dish square, as follows:

| Density | Character |
|---|---|
| 0 | . |
| 1 | ! |
| 2 | X |
| 3 | # |

No other characters may appear in the output.

## Sample Input:

```
2
0 1 1 1 2 1 0 -1 -1 -1 -2 -2 -3 -3 -3 -3
3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

## Sample Output:

```
##!.................
#!..................
!...................
....................
....................
....................
....................
.........!..........
........!#!.........
.......!#X#!........
........!#!.........
.........!..........
....................
....................
....................
....................
....................
....................
....................
....................
```

# Problem E: Algernon's Noxious Emissions

One of the greatest alchemists of the lower Middle Renaissance, Algernon da Vinci (one of Leonardo's lesser-known cousins), had the foresight to construct his chemical works directly over a fast-running stream. Through a series of clever pipes and sluices, he routed portions of the stream past each of the tables where his alchemists prepared their secret brews, allowing them to dispose of their chemical byproducts into the waters flowing by the table.

As Algernon's business grew, he even added additional floors to his factory, with water lifted to the higher floors by treadmill-powered pumps (much to the dismay of the apprentices who found themselves assigned to pump duty). The pipework for the entire disposal system became quite complex. It was even rumored by some that the pipes actually circled back in some places, so that a particularly odorous compound flushed away from one table might return to that very same spot a few minutes later.

All was not well, however. Algernon's factory suffered from a series of mishaps, minor explosions, gas clouds, etc. It became obvious that chemicals dumped at one table might react violently with other chemicals dumped from another table downstream. Algernon realized that he needed to trace the possible chemical flows through his factory.

Write a program to aid Algernon in this task. To preserve the secrecy of the chemical processes that are Algernon's stock in trade, all chemicals will be identified by a single upper-case letter. All tables are identified by positive numbers in the range 1…N, where N is the number of tables.

## Input Format:

Line 1:
# of work tables, integer (henceforth referred to as N). N < 50

Lines 2…N+1
For each table:

- a list of chemicals dumped into the stream at that table, followed by
- a list of chemicals that, if they appeared at that table, would be harmlessly neutralized by the reactions at that table, allowing no further trace of that chemical to flow downstream (we will assume that the rate of work at each table can be adjusted as necessary to guarantee total neutralization of whatever amount of these chemicals arrive from upstream).

Each of these lists is given as a series of upper-case alphabetic characters. The only exception is that a special list, consisting of a single '.' character, will be used to denote an empty list. The two lists are separated from one other by one or more blanks. The same chemical will never appear in both lists.

Lines N+2…?
These lines provide a description of the pipeworks. Each line contains a pair of integers in the range 1…N, separated by one or more blanks:
I J
meaning that the table number I is upstream of table number J—anything dumped into the stream at table I or that arrives in the stream at table I and is not neutralized can then be counted on to arrive at table J.

No (I,J) pair will be listed more than once, but the pairs may occur in any order. I and J will never be the same number.

The end of input is signaled by a pair of zeros:
```
    0 0
```

Note that if a table only receives water directly from the stream entering the building, that table will never occur in the second position of a pair. Similarly, any table that discharges only into the stream leaving the building will never occur in the first position of a pair.

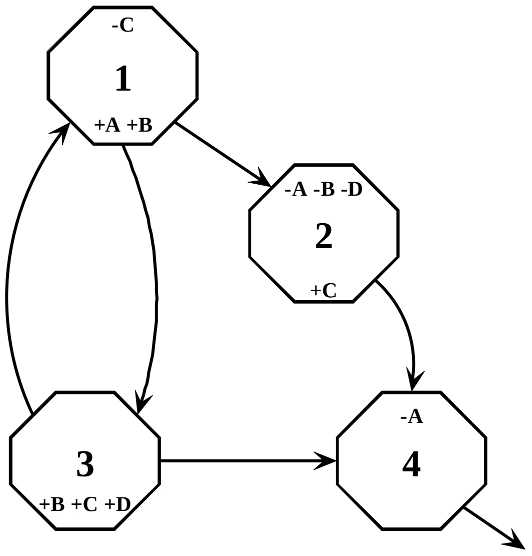
## Output Format:

There will be N lines of output, one for each table, in the same order as they appeared in the program input. Each line will contain the list of chemicals that can be expected at that table's output. This list will be printed as a (possibly empty) list of upper-case alphabetic characters between two colons (:). No empty spaces should be printed on the line. The characters in the list should be sorted in alphabetic order.

## Sample Input:

For the figure at the right, an input file would be:

```
4
AB C
   C BDA
BCD .
. A
1 2
 2 4
3 1
1 3
3 4
0   0
```



## Sample Output:

```
:ABD:
:C:
:ABCD:
:BCD:
```

## Problem F: FDNY to the Rescue!

The Fire Department of New York (FDNY) has always been proud of their response time to fires in New York City, but they want to make their response time even better. To help them with their response time, they want to make sure that the dispatchers know the closest firehouse to any address in the city. You have been hired to write this software and are entrusted with maintaining the proud tradition of FDNY. Conceptually, the software will be given the address of the fire, the locations of the firehouses, street intersections, and the time it takes to cover the distance between each intersection. It will then use this information to calculate how long it takes to reach an address from each firehouse.

Given a specific fire location in the city, the software will calculate the time taken from all the fire stations located in the city to reach the fire location. The list of fire stations will be sorted from shortest time to longest time. The dispatcher can then pick the closest firestation with available firefighters and equipment to dispatch to the fire.

## Input Format:

Line 1:

# of intersections in the city, a single integer (henceforth referred to as N) N<20

Lines 2 to N+1:

A table (square matrix of integer values separated by one or more spaces) representing the time taken in minutes between every pair of intersections in the city. In the sample input shown below the value "3" on the 1st row and the 2nd column represents the time taken from intersection #1 to reach intersection #2. Similarly the value "9" on the 4th row and the 2nd column represents the time taken from intersection #4 to reach intersection #2.

A value of -1 for time means that it is not possible to go directly from the origin intersection (row #) to the destination intersection (column #). All other values in the table are non-negative.

Line N+2:

An integer value n (<= N) indicating the intersection closest to the fire location followed by one or more integer values for the intersections closest to the fire stations (all on one line, separated by one or more spaces) will follow the input matrix.

**Notes on input format:**

1. The rows and columns are numbered from 1 to N.
2. All input values are integers
3. All fire locations are guaranteed reachable from all firehouses.
4. All distance calculations are made from the intersection closest to each firehouse to the intersection closest to the fire.

## Output Format:

Line 1:

A label line with the headings for each column, exactly as shown in the example.

Line 2 onwards (one line for each fire station):

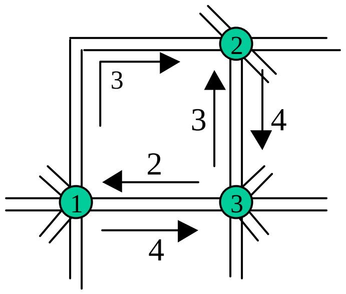
A sorted list (based on time) showing the fire station (origin), the destination site, time taken and a complete shortest path of nodes from the originating fire station to the fire location.

**Notes on output format:**

1. Columns are tab separated.
2. If two or more firehouses are tied in time they can be printed in any order.
3. If more than one path exists that has the same minimal time for a given location & firehouse, either one can be printed on the output.
4. If the fire location and the fire station locations happen to be the same intersection, the output will indicate that the origin and destination have the same intersection number, the time will be "0" and the nodes in the shortest path will show just one number, the fire location.

## Sample Input:

```
   6
   0    3    4   -1   -1   -1
  -1    0    4    5   -1   -1
   2    3    0   -1   -1    2
   8    9    5    0    1   -1
   7    2    1   -1    0   -1
   5   -1    4    5    4    0
   2    4    5    6
```



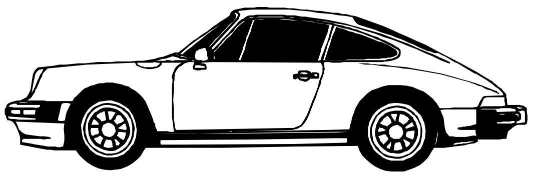
A partial map of the city representing the time taken between intersections

In the above input the last line indicates that “2” is the location of the fire and “4”, “5” and “6” are the intersections where fire stations are located.

## Sample Output:

```
Org  Dest Time Path
5    2    2    5    2
4    2    3    4    5    2
6    2    6    6    5    2
```

# Problem G: For the Porsche



The Cash Cow Consulting Company is challenging the Vice Presidents to increase the profitability of their departments. In an effort to provide proper incentive, the Vice President whose department has the highest Profitability Index (PI) will win a brand new Porsche. The contest rules are as follows:

- The winning department will have the maximum Profitability Index (sales / development cost)
- Each department must stay within the minimum and maximum cost range.
- In the case of equal profitability indexes, the higher profit margin will win (sales – development cost).
- If two departments are still tied, the winning department will develop the smaller number of features.
- If two departments are still tied, the winning department will satisfy the most customers.

Mike Miser is still driving his high school moped and has determined this is his chance to upgrade. He has instructed the engineering department to determine what it will cost for each feature to be developed. He then instructed the sales force to determine what features each customer requires, and what sales that will generate. (To make a sale to a customer all features required must be provided).

Mike will then determine which feature combination his division should complete to maximize his chances of winning the contest.

## Notes:

1. Because of the type of product the Cash Cow Consulting Company creates, the production costs are negligible, and do not need to be considered. Only the development costs should be considered.
2. The tie breakers listed will result in the selection of exactly one feature set.
3. At least one feature set will satisfy the requirements.
4. The Profitability Index should be rounded to three decimal places. The values 3.4566 and 3.4574 will be considered equal.

## Input Format:

All input will be positive integers.

The first line of input will indicate the number of data sets.

The first line of each data set will contain 4 integers separated by white-space. In order they are the minimum cost, maximum cost, number of potential features (N) and number of potential customers (M). N and M will be no larger than 20.

The next N lines (one for each feature) indicate the cost of each feature.

The next M lines will contain the following (one line for each customer):

Number of required features Feature number (for each required feature) Total Sales for that customer.

For instance, if a given customer wanted 3 features, number 1,2 and 5 and would provide sales of 50, the line would read: "3 1 2 5 50"

The next data set, if more remain, will begin on the next line.

## Output Format:

The first line of output for each data set should indicate which Feature Set is being considered. These should print "Feature Set N" where N is the feature set number, counting from 1.

The next line of output for each data set should indicate the profitability index to 3 decimal places.

The next line of output for each data set should indicate the sales dollars

The next line of output for each data set should indicate the cost

The next line of output for each data set should indicate which features are implemented. The first feature is feature number 1. They must be listed in order, white-space separated.

The final line of output for each data set should indicate the customers who were satisfied. The first customer is customer #1. They must be listed in order, white-space separated.

No extra output should appear.

## Sample Input:

```
1
100 2000 7 6
250
350
400
250
250
250
500
4 1 4 5 6 4000
4 1 4 5 6 500
4 1 4 5 6 60
3 1 4 5 7
4 1 2 3 5 5
4 1 2 3 7 6
```

## Sample Output:

```
Feature Set 1
4.567
4567
1000
1 4 5 6
1 2 3 4
```

# Problem H: Oh, Those Achin' Feet

In recent days, a number of people have been injured after being pushed off the sidewalks due to overcrowding. City Hall is interested in figuring out how much pedestrian traffic its sidewalks receive every day. The results of this study will be used to determine whether the city needs to fund more sidewalks. The city has surveyed various buildings in several blocks to determine the traffic patterns they generate. Your job is to take this survey data and convert it into sidewalk utilization information.

Your program will read in the size of the map and a map of several city blocks. Buildings, streets, and building entrance/exits will be marked on the map. You will also be given a list of pedestrian load between several pairs of exits and entrances. Your program will determine the paths used by pedestrians between each source and destination, add up the total pedestrian load from all paths using each street, and output a table of the total pedestrian load on each square.

**Notes:**

- The map is divided into squares. Each square of the map can be a street square, a building square, or an entrance/exit square. An entrance/exit square serves as both entrance and exit for that building. There will be no more than 90 street squares in the map.
- People will always follow the shortest path between their origin and destination. No shortest path will exceed 75 squares.
- If there are multiple equal-length shortest paths, the load will be divided equally amongst the paths. For shortest paths, there will be fewer than 50000 equal-length path combinations.
- If a building entrance/exit has multiple sides facing a street (for example, a corner of a building), the pedestrians may enter or exit through any street-facing side.
- All movement will be strictly N, E, S, or W. No diagonal movement is permitted.
- Pedestrians cannot move through buildings or off the edge of the map.
- For convenience, you may ignore the fact that each street section may have two sidewalks.
- Traffic load is not applied to the actual exit/entrance squares themselves.
- If an origin and destination are adjacent on the map, pedestrians may move directly between them. In this case, there is no resulting load placed on any portion of the map because no streets are used.

## Input format:

Line 1: X Y
 X is the number of columns in the map, Y is the number of rows. Each is a positive integer less than 20.
Line 2-(Y+1):
 Each line contains exactly X symbols indicating the contents of that square on the map. The symbols are:
  `X`: building, non-entrance/exit
  `.`: (period) street
  {`A-O`}: letter indicating exit/entrance. Each letter may occur at most once.
Lines (Y+2)-?:
 Each line indicates a pedestrian route and specifies a source, destination, and pedestrian load. Source and destination will each be a letter {`A-O`} with no spaces in between. The load factor will be a nonnegative integer, separated from the destination by whitespace. Source and destination will never be equal. At most 25 routes will be given. There will be a valid path in the map for each requested route.

The file will terminate with the line:
`XX 0`

## Output format:

The output consists of Y lines, each with X space-separated fields indicating the load factor. Each load factor is printed to two decimal places with 3 spaces for integer digits (C 6.2 format).

## Sample Input:

```
4 4
....
A.X.
XXX.
B...
AB 2
BA 1
XX 0
```

## Sample Output:

```
  1.50   3.00   3.00   3.00
  0.00   1.50   0.00   3.00
  0.00   0.00   0.00   3.00
  0.00   3.00   3.00   3.00
```