Problem 1: Court Ordering

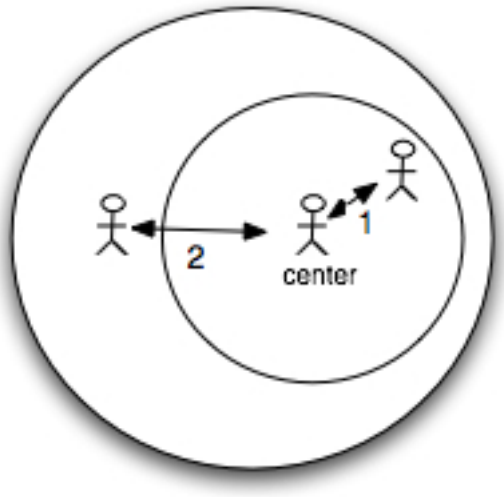
The president is in a predicament: should he select a judge with different opinions of the judges currently on the court, or should he stick with a judge like those currently on the court? Picking a candidate that shares the opinions of the existing judges on the court results in a harmonious court; on the other hand, selecting a candidate with different opinions results in a more diverse court.

Judges are trained at law schools, and the professors and students they work with at the law school influence their thinking, and ultimately their opinions: think of a law school as a set of persons sharing common thoughts. At different times a person may belong to a different law school, even several at one time, and thus share some more common thoughts.

The thoughts are what matters. If you pick a judge, you pick the thoughts of the members of whatever law schools he or she was in — call this the first circle. Each school, however, might well share the thoughts of the additional schools all of its members were in — the second circle. You can continue to form additional circles as long as there are more members to be linked.

Now consider a court. It's thoughts depend on the thoughts of all the law schools which the court's judges attended — the union of their first circles. Additionally, you can consider the union of the second circles — the additional schools all those school's members attended — or even further.

You can define a distance for each member of a circle (or union of circles): assign distance 0 to the member (or entire court) at the center, distance 1 to the members in the first circle (or in the union of first circles), and so on out until distance *n* to the remaining persons in the *n*'th circle (or union) that is considered.



For a candidate, this defines distance in *n* circles around the candidate. For a court, this defines distance in *n* unions of circles around the court. Consider a person *x* who appears in both, a candidate's *a*'th circle and a court's *b*'th union of circles: *x* has distance *a* from the candidate and *b* from the court. *x* contributes

a times *n-b* to the candidate's diversity, which is defined to be the sum over the contributions of all such persons *x*.

In the example given below, `whale`'s first circle contains `cato`, `pablo`, and `nathan`, who have distance 0, 1, and 1 from the court. Because of `pablo`, `whale`'s second circle contains `jeffy` and `wanton` with distance 1 and 2 from the court. The example considers two circles, therefore, `cato` contributes 2, `pablo` and `nathan` each contribute 1, and `jeffy` contributes 2 to `whale`'s diversity of 6.

There is, however, a dark horse rule: If a candidate's circles and the court's circles do not have any members in common, the candidate is called a dark horse and cannot be considered for the court position.

You are asked to write a program that will help the president select a candidate for the court. Input to your program consists of a line with the court, a line with candidates, a line with a number indicating how many circles (and unions) to consider, a line with the number of law schools, and finally the sequence of law schools, one per line. Each line for court, candidates, or law school consists of a sequence of names, separated by white space. Names themselves do not contain white space and comparison ignores the case of characters.

Your program will output one line labeled `harmonious` with the names of all candidates for a harmonious court that have the same smallest diversity value followed by that value, a second line labeled `diverse` with the names of all candidates for a diverse court that have the same largest diversity value, again followed by that value, and finally a line labeled `dark horse` with the names of all candidates subject to the dark horse rule, if any.

Sample Input:

```
cato cesar cicero
wanton wishful whopper whale
2
5
cato pablo nathan whale
cesar penelope jeffy wishful
cicero hito mike
pablo wanton jeffy
whopper hannah
```

Sample Output:

```
harmonious whale wishful 6
diverse wanton 18
dark horse whopper
```

Problem Number 2: Bacteria Colonies

Biologists use bacteria to clone DNA for sequencing. These bacteria are injected with the DNA that needs to be cloned and then allowed to reproduce on a large plate. As these bacteria reproduce they form colonies. Only bacteria from healthy colonies will be used for sequencing.

Your program will be given a pixelized snapshot of a plate, *W* pixels wide and *H* pixels high. Pixel (0,0) is in the upper left hand corner of the snapshot, and pixel (*W*-1,*H*-1) is at the bottom right hand corner of the snapshot. Each pixel in the snapshot is either a 0, which indicates no bacteria present at that location, or 1 indicating the presence of bacteria.

We are interested in finding colonies consisting of healthy bacteria that grow in a very specific way. Bacteria in these colonies have the property that when they grow, bacteria from every cell in the entire colony will grow into each of the 4 adjacent cells that have either the same row, or column as the original cell. Therefore a healthy colony consists of 5 or more cells, as illustrated by the following two examples:

```
  *          *
 ***        ***
  *        *****
            ***
             *
```

The following are examples of unhealthy colonies:

```
    *
***    *  *      ****
    *         ****
```

Furthermore, to avoid cross contamination, colonies must not be too close to the edge of the plate or each other. The diameter of a healthy colony is the number of pixels in the largest row or column of the colony. Every cell on the boundary of a healthy colony must be at least *diameter* pixels away, measured horizontally and vertically, from any edge of the plate and any other pixel that contains bacteria outside of the colony.

The input to your program will consist of a line containing two positive integer values separated by blanks; *W* the width, and *H* the height of a plate. The next H lines of input will each contain W zeros or ones indicating the status of the corresponding pixel (i.e. 0 meaning no bacteria present, 1 meaning bacteria is present), starting with row 0 and ending with row *H*-1.

Your program will determine which colonies on the plate are healthy. It will produce as output the count of the number of healthy colonies, followed by the coordinates of the center pixel of each healthy colony (if any).

Sample Input:

```
15 15
000000001000000
000000000100000
000000000010000
000000000001000
000000000000100
000000010000010
000000111000001
000001111100000
000000111000000
000000010000000
000000000000000
000000000000000
000000000000010
000000000000111
000000000000010
```

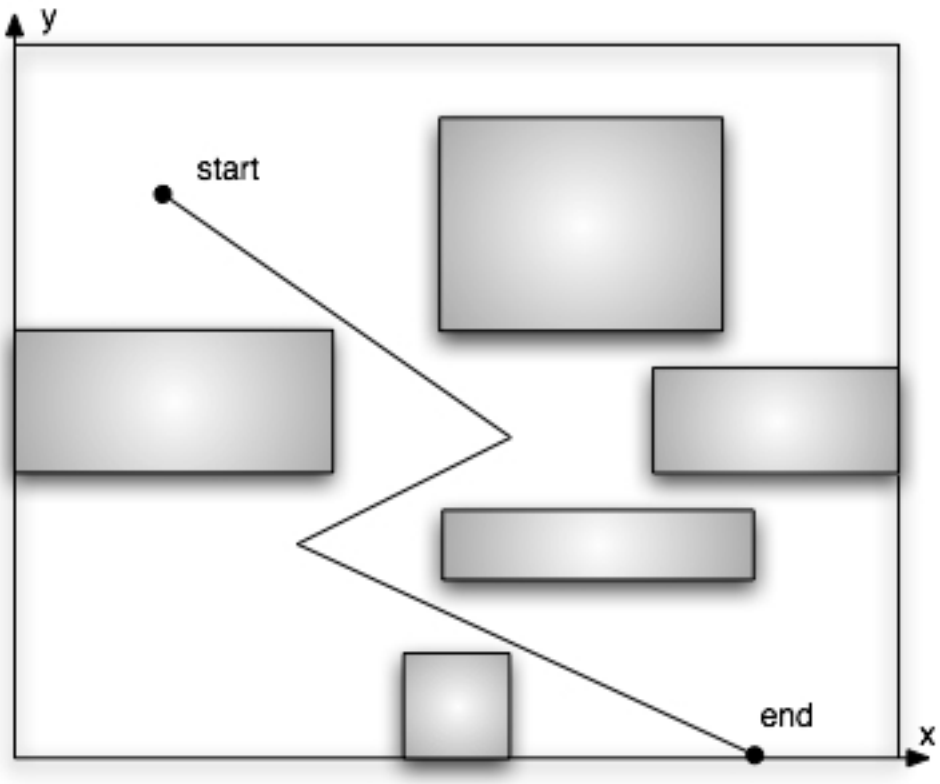
Sample Output:

```
Number of healthy colonies:  1
Colony 1: 7, 7
```

Problem Number 3: James' Room

James is not known for keeping his room very clean. Scattered about his room are rectangular boxes of computer parts, which makes it very difficult to get from one location in the room to another. Although James is not very neat, he is very careful about the way in which the boxes are placed in the room. The boxes are placed in such a way that the sides of the boxes facing a wall are parallel to the wall.

The challenge that James faces is to find a path, consisting of a specific number of straight line segments which end on points with integer coordinates that gets him from one location in his room to another without stepping on or over any boxes. The diagram below shows a possible configuration of James' room containing five boxes, and a path consisting of three line segments that lead from start to end.



You are to write a program that given the dimension of James' room, his starting location and desired destination, the number of line segments in the path, and the location of the boxes, prints the path if one exists.

The first line of input to your program will contain seven integers X, Y, X_S, Y_S, X_D, Y_D , S that specify the extent of the room in x- and y-direction, the coordinates of the location where James will start his walk, the coordinates of the location of his

destination, and the number of straight line segments in the path. *X* and *Y* will be positive, start and destination will be within the boundary of the room, and *S* will be 2 or more.

The second line of input will contain a single positive integer, *N*, that gives the number of boxes in the room. Each of the next *N* lines will contain four positive integers X_1, Y_1, X_2, Y_2 that specify the location of two opposite corners of a box and will be within the boundary of the room. The boxes will not overlap, but may touch (and James cannot touch a box).

If a path exists your program will print integer coordinates of the $S - 1$ intermediate points, in order, for a path containing *S* straight line segments that starts at location X_S, Y_S, and ends at location X_D, Y_D. If such a path (with integer coordinates) does not exist the program will print the word `impossible`.

Sample Input:

```
25 20 4 16 21 0 3
5
0 8 9 12
12 12 20 18
18 8 25 11
12 5 21 7
11 0 14 3
```

Sample Output:

```
14 9
8 6
```

Problem Number 4: Monkey Business

A technician has been training the monkeys used in the lab where she works to open and close doors. Next to the lab is a long hallway that contains a number of offices which initially have closed doors. The offices in the hallway are numbered starting at 1 and ending at *N*, where *N* is the number of offices in the hallway. Interestingly the number of monkeys in the lab is exactly equal to the number of doors in the hallway.

At night the technician lets the monkeys out of their cages one at a time for exercise. The first monkey that is let out of the cage runs down the hallway and opens every door (i.e., 1, 2, 3, 4, ...). The second monkey runs down the hallway and closes all of the even numbered doors (i.e., 2, 4, 6, 8, …). The third monkey runs down the hallway and examines every third door (i.e., 3, 6, 9, 12, …). The monkey will open any closed door that it examines, and close any open door that it examines. The fourth monkey examines every fourth door, and so. This process continues until all of the monkeys have been let out of their cages and allowed to run down the hallway — the last monkey will only examine the last door.

It would be nice to know which doors are left open at the end of the night. The technician has decided to write a grant to fund this work and has asked you for help. She needs you to write a program, that will be given as input the number of doors in a hallway — assuming that there are as many monkeys as there are doors, and that all of the doors are initially closed. Your program will output a list of numbers, one per line, that identify the doors that will be open after all of the monkeys have been let out of their cages.

For example, given a hallway that contains 5 doors (and 5 monkeys), after all the monkeys have been let out of their cages doors 1 and 4 will be open.

The input to your program will consist of a single positive integer value that specifies the number of doors in the hallway. Your program will produce as output a list that specifies the doors that are open after all of the monkeys have been let out of their cages.

Sample Input:

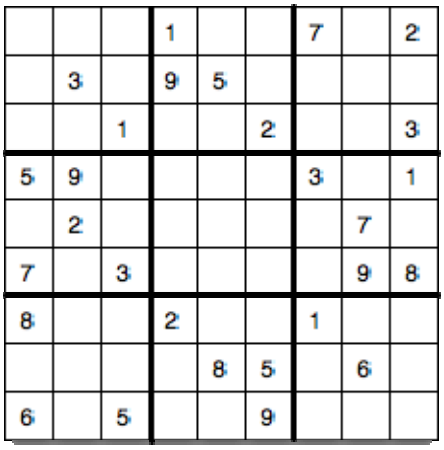
```
5
```

Sample Output:

```
1
4
```

Problem Number 5: Sudoku

In a Suduko puzzle you are presented with a 9 x 9 grid that is divided into smaller 3 x 3 non-overlapping sections or “regions”. Some cells in the grid will be pre-populated with the digits 1 to 9. An example of a Sudoku puzzle is shown below:



			1			7		2
	3		9	5				
		1			2			3
5	9					3		1
	2						7	
7		3					9	8
8			2			1		
				8	5		6	
6		5			9			

Your task is to fill the remaining empty cells so that each row, column and 3x3 region contains one instance of every digit from 1 to 9.

Write a program that reads the 81 digits for the grid cells in row order, separated by white space, where a 0 indicates that the cell is not yet filled and any other digit indicates a digit already placed into the cell. Your program will produce as output either a filled-in grid or the word `impossible`. You may assume for grids for which a solution exists, there is only one valid solution. The input representing the puzzle shown above is given below.

Sample Input:

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

Sample Output:

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

Sample Input:

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

Sample Output:

```
Impossible
```

Problem Number 6: The pPod

Orange computers, a fierce competitor of another well-known fruit-oriented computing company, has announced plans to develop the pPod. The pPod is an MP3 player that uses a revolutionary software system to manage its memory. Currently all pPods come with 4096 megabytes of storage.

Songs are placed into the pPod's memory contiguously (i.e., one after another) starting at location 0. When a song is deleted, the space allocated to that song is marked as unused. The memory in the pPod can be viewed as a contiguous sequence of memory segments that either contain songs or are unused. Each memory segment is identified by a unique integer value that is equal to the address of the first byte in the segment.

When adding a song to the pPod, the memory segments are scanned, in ascending order by identifier, looking for the first unused segment that is large enough to hold the song. If the segment is larger than the song, the segment is broken up into two pieces, one for the song and one for the remaining unused memory. There is memory recombination but no shifting of songs.

You are to write program that will test your memory management system using a simple text interface. Your program will accept the following commands (note that all sizes are given in megabytes):

Command	Syntax	Description
Add	a *song_title size*	Add the specified song title with the given size to the pPod. The song will be added if the title has not already been added to the pPod and a memory segment of a suitable size can be found.
Delete	d *song_title*	Remove the specified song from the pPod.
List	l *song_title*	If the song title is in the pPod, display the starting address of the song in memory, the size of the song, and the title in the format shown below: *aaaa*:*ssss* - - *song_title* where *aaaa* is the address, *ssss* is the size and *song_title* is the song title.
Memory	m	The segments are listed in ascending order by identifier. Occupied segments use the same format as for the list command, unused segments are displayed without a title.
Exit	e	Terminate the program

Song titles do not contain any spaces. All string comparisons are case insensitive (i.e., Song, song, SONG are all the same). If any error occurs while processing a command, your program will print “??”, and read the next command.

A sample run of the program is given below. Lines printed in **bold** represent input to the program.

Sample Run:

```
a aqualung 1024
a thick as a brick
??
a aqualung 1024
??
a thick_as_a_brick 4096
??
a bouree 512
a songs_from_the_woods 512
m
0:1024 -- aqualung
1024:512 -- bouree
1536:512 -- songs_from_the_woods
2048:2048
d bouree
m
0:1024 -- aqualung
1024:512
1536:512 -- songs_from_the_woods
2048:2048
l passion_play
??
a passion_play 1024
m
0:1024 -- aqualung
1024:512
1536:512 -- songs_from_the_woods
2048:1024 -- passion_play
3072:1024
l passion_play
2048:1024 -- passion_play
d songs_from_the_woods
m
0:1024 -- aqualung
1024:1024
2048:1024 -- passion_play
3072:1024
e
```

Problem 7: Spaced out

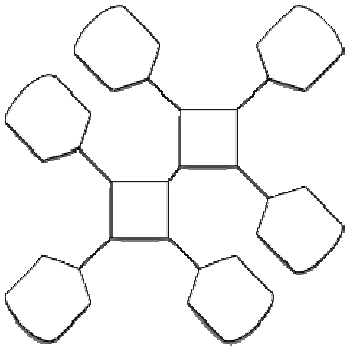
FALSE, the Flatland Agency for Lustful Space Exploration, has just issued their TRUE proposal — Terrific Rooms Uniting Easily, a multi-billion-flop to be (recall that flop is the unit of currency in this charming two-dimensional country).

TRUE's come in two flavors: Couches, each consisting of a dome with a single tunnel, will be used as private spaces for the FALSE astronauts. Hives, each consisting of a dome with four tunnels, will connect couches and other hives and will serve as laboratories and conference rooms.

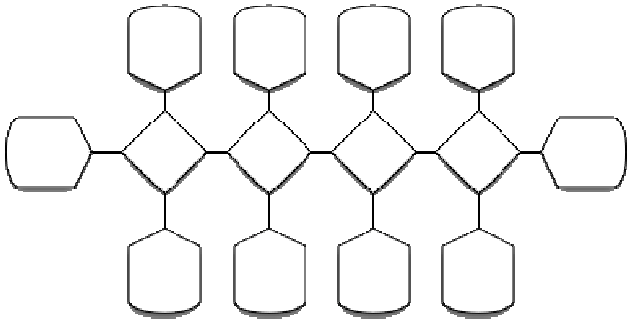
Of course, connections are made only from one tunnel to another, and no tunnel may be left unconnected to avoid accidents in the void. Unfortunately, FALSE astronauts tend to get lost easily; therefore, the TRUE proposal explicitly forbids any tunnel connections that would form cycles or, even worse, connect a hive to itself.

Here are a few TRUE stations:

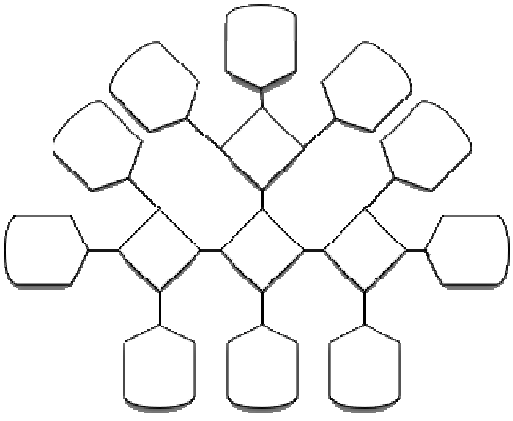
(a)



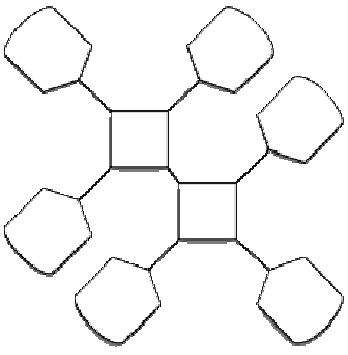
(b)



(c)



(d)



Although (b) and (c) contain the same number of hives, they are still different from each other. Of course, they are both different from (a) which consists of two hives. (a) and (d) are equal, but for position.

Not unlike other government agencies, FALSE believes in standards, especially in multi-billion-flop standards. They are asking that you create a LOUSY[1] program which computes the number of different TRUE stations that can be constructed from a given number of hives.

The input to your program is a positive integer value designating the number of hives to be used for one station. Your program will print a single integer value — the number of different TRUE stations that can be created from this many hives.

Sample Input:
2

Sample Output:
1

Sample Input:
9

Sample Output:
35

[1] Limit On Uniting Space Yokes

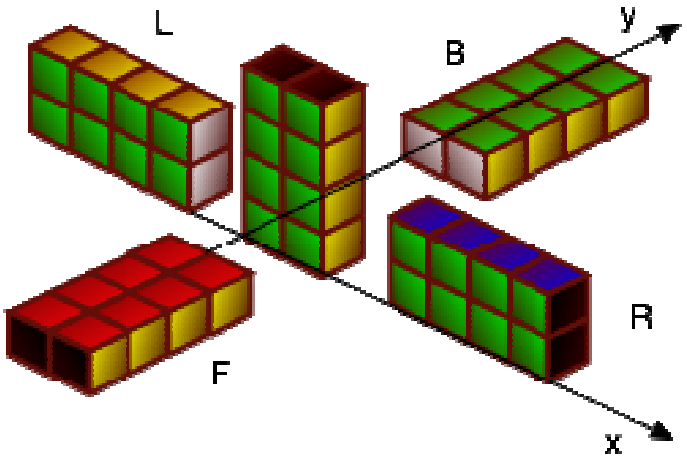
Problem 8: Rolling Cubes

A wooden block is standing on a large board with 1” squares. The block is one inch deep, 2 inches wide and 4 inches tall. The block is sitting on exactly two squares. The top of the block is black, the bottom is white, the front is green, the back is red, the left face is blue, and the right face is yellow.

The block is moved to a new location on the board by rolling it. The block is rolled in such a way that one edge is always in contact with a line of the board at all times. The block never slides.

There are four basic operations that can be performed on the block:

F: Roll the block forward
B: Roll the block backwards
L: Roll the block to the left
R: Roll the block to the right



The diagram shows the original position of the block (center) and the aspect *but not the exact board location* of the block after each of the four basic operations.

Write a program that when given a string containing several letters from the set F, B, L, and R, calculates the resulting position of the block following the sequence of basic operations specified in the string and prints out the new location of the front left corner of the block relative to the original location shown above and prints out the colors on the front, left, and top of the block.

Sample Input:

```
FRBL
```

Sample Output:

```
x: -2, y: 0, front: yellow, left: black, top: red
```