Problem No. 1 –Anthrax Lab Studies
Executable Program : PROG1.EXE, PROG1.CLASS
Source Program: PROG1.CPP, PROG1.JAVA, PROG1.C, PROG1.PAS
Input file: PROG1.IN Output file: PROG1.OUT

Anthrax, also called Splenic Fever, Malignant Pustule, or Woolsorters' Disease, is caused by Bacillus anthracis, an organism that under certain conditions forms highly resistant spores capable of maintaining its virulent effect for many years in contaminated soil and other materials.

Anthrax was the first human disease that was demonstrated to be caused by a specific microrganism. It was also the first infectious disease against which a bacterial vaccine was found to be effective (Louis Pasteur in 1881). These discoveries led to the origin and development of bacteriology and immunology.

Interest has reborn around the study of how infected cells reproduce and die. A hypothesis states that cell reproduction can be modelled as a two-dimensional infinite grid of square cells like chessboard cells but in Practical lab studies its defined as finite grid. In this grid, each cell is considered to be alive or dead. A living cell is represented as a mark on its position in the grid. Each cell has up to 8 neighbouring cells (up, down, left, right, upper-right, upper-left, lower-right and lower-left). Cells located in the border of the grid have smaller number of neighbours.

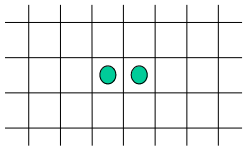
The model starts with an Initial Generation (G_0) consisting of some living and death cells in the Grid. After reproduction rules are applied to every Cell, a new Generation G_1 is obtained. New generations are always created using the information of the previous generation. That is Generation G_i is created based on information of Generation G_{i-1}

The reproduction rules that are used in the model are:

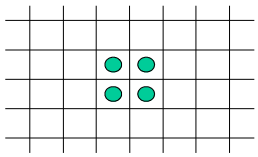
1. A dead cell with exactly three living neighbours becomes a living cell (birth).
2. A living cell with cero or one living neighbours dies (loneliness).
3. A living cell with two or three living neighbours stays alive (survival).
4. A living cell with four or more living neighbours dies (overcrowding).

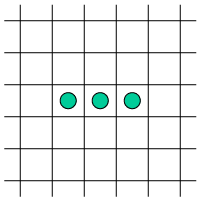
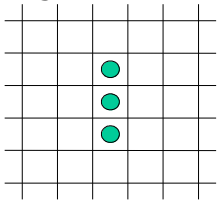
In all other cases, a cell dies. Existing death cells remain death (overcrowding or loneliness).

EXAMPLES.
This initial generation will evolve in a generation where all cells die.



This initial generation yields generations that will always be the same as the initial.



If the initial generation is  the next generation will be  the following generations will be alternating between these two generations.

In more complex cases, it is impossible to look at a starting position (or pattern) and see what will happen in the future. The only way to find out is to follow the rules

INPUT
The input of the program will be a text file, (located in the same path as the executable). The input will contain data for several runs, each run will have the following information:

The first line has two numbers indicating the X and Y dimensions of the grid. (X number of columns, Y number of rows). A 0 in this first line indicates end of run.
The second line is one number representing the number of generations that must be obtained.
The third line is a number, that specifies the number of cells

that have a microrganism.
Starting in the forth line, there will be the coordinates of each microrganism, one per line with format x, y (x is column number and y is row number). The row and column numbers start at 1.

OUTPUT

The output is a text file (located in the same path that the program). In this file you have to show the different generations starting with the Initial Generation. A Generation is represented as a matrix showing Living microrganisms as an "*" (asterisk) and death cells as a "0" (cero).

Before printing the Initial Generation you should output the study number. Before the Initial Generation write the string "Initial Generation". Before each new generation write "Generation x", where "x" is the generation number.

SAMPLE INPUT FILE

```
15 10
5
3
5 5
5 6
5 7
0
```

SAMPLE OUTPUT FILE

Study 1
Initial Generation

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 * 0 0 0 0 0 0 0 0 0 0
0 0 0 0 * 0 0 0 0 0 0 0 0 0 0
0 0 0 0 * 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Generation 1

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 * * * 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Generation 2

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 * 0 0 0 0 0 0 0 0 0 0
0 0 0 0 * 0 0 0 0 0 0 0 0 0 0
0 0 0 0 * 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Generation 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

Generation 4

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 * 0 0 0 0 0 0 0 0 0 0
0 0 0 0 * 0 0 0 0 0 0 0 0 0 0
0 0 0 0 * 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Generation 5

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 * * * 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Problem No. 2 – Postfix
Executable Program : PROG2.EXE, PROG2.CLASS
Source Program: PROG2.CPP, PROG2.JAVA, PROG2.C, PROG2.PAS
Input file: PROG2.IN Output file: PROG2.OUT

A junior high school has found that many of their students have problems with the conceptualisation of mathematical expressions evaluated in postfix. You were contacted by the school to develop a program that could evaluate expressions in postfix, so that the students can practice with it helping them to keep the objectives of the class.

The program was requested by the professor of Mathematics I. For this reason the program should only handle operations with two operands and one operator, real numbers and the four basic operations: addition, subtraction, multiplication and division. An addition request was made. The program could only remember two operands.

INPUT
The input of the program will be a text file containing the data for several runs. Each line will have one expression. Each expression will have a valid structure. Each line could be up to 100 characters. The program should stop after processing all the expressions.

OUTPUT
The output is a text file. In this file you have to show the result of each expression, one result per line. It should have a four digit precision.

SAMPLE INPUT FILE
```
123 1254 + 1.254 * 45.87 /
5 4 +
5 4 + 1 + 2 + 12 -
3 2 * 1 - 2 / 3.1415 +
```

SAMPLE OUTPUT FILE
```
37.6446
9
0
5.6415
```

Problem No. 3 –WEB Reservation Application
Executable Program : PROG3.EXE,PROG3.CLASS
Source Program: PROG3.CPP, PROG3.JAVA,PROG3.C, PROG3.PAS
Input file: PROG3.IN Output file: PROG3.OUT

The travel agency where you work, asked you to modify the Web Reservation Application Page to include the following functionality

FUNCTIONALITY

Given a departing city C and an airline A, determine all the cities that can be reached from C using only airline A.

The number of airlines is n (1<=n<=25) and are identified as 0 to n-1. The number of cities is m (2<=m<=100) and the cities are identified as 0 to m-1.

INPUT

The first line of the input file contains two positive integers: The number of existing cities (m) and the number of existing airlines (n).
The next lines contain several positive integers. Each line contains the departing city number, an airline number and the city numbers for all the cities that you can get to from the departing city using single flights by the given airline.
The last line contains only two positive integers: The number of a departing city and the number of the airline that you want use.

OUTPUT

The output of your program will be a file with a single line of positive ordered integers. Based on the information of the last line in your input file, the output line contains the city number of all the cities that you can reach from the departing city , using only the given airline. Do not include in the output, the departing city number.
When you can´t get to any city from the departing city using the given airline, the output line should contain the next message : "No hay destinos posibles por esta linea"

SAMPLE INPUT FILE

```
17 3
1 1 2
1 0 6
1 2 5
2 1 3
2 0 7
2 2 6
3 1 4
3 0 8
3 2 7
4 2 8
5 1 6
5 0 10
5 2 9
6 1 7
```

```
6 0 11
6 2 10
7 1 8
7 0 12
7 2 11
8 2 12
9 1 10
9 0 14
9 2 13
10 1 11
10 0 15
10 2 14
11 1 12
11 0 16
11 2 15
12 2 16
13 1 14
14 1 15
15 1 16
1 0
```

SAMPLE OUTPUT FILE

```
6 11 16
```

SAMPLE INPUT FILE

```
100 100
1 0
```

SAMPLE OUTPUT FILE

```
No hay destinos posibles por esta línea
```

SAMPLE INPUT FILE

```
10 10
0 1 1 4
0 2 8
0 6 3 8
0 7 1
0 9 2 3 4
1 0 4
1 2 3 9
1 5 4
1 6 8
1 7 6
1 9 6
2 0 0 5 6
2 2 6
2 5 1 9
2 7 4 9
2 8 4 8 9
3 0 7
3 4 0
```

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

SAMPLE OUTPUT FILE

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

Problem No. 4 –Cheap Fence
Executable Program : PROG4.EXE, PROG4.CLASS
Source Program: PROG4.CPP, PROG4.JAVA, PROG4.C, PROG4.PAS
Input file: PROG4.IN Output file: PROG4.OUT

DESCRIPTION

A farmer has decided to surround his garden in order to protect his fruits from the appetite of his neighbors. This man does not have a lot of money, but he is a very innovative person. He has found the possibility of reducing costs by using trees from his property as fence posts. He found that in order to maximize the area to be enclosed by the fence, the trees to be used as fence posts must be selected carefully avoiding the creation of inward angles in the fence. Also, the selection of tree posts must ensure that the fence will enclose all the trees that are in the property.

PROBLEM

Given a set of trees (each tree is identified by a unique tree identifier) and the position of each tree in the property, write a program that obtains the following information:

a) The minimum number of trees required to enclose all the trees in the property
b) The sequence of trees that must be used as tree posts.

INPUT

The first line of the input file contains an integer value (N) representing the number of trees available in the garden. $1 < N < 100$
The following N lines are triplets representing the tree id (integer number) and the position of the tree (X,Y coordinates) in the garden (the coordinates are real numbers).

OUTPUT

The first line of the PROG4.OUT file, displays the number of trees that must be used as fence posts. The next line displays the sequence of tree identifiers that will be used as fence posts.

EXAMPLE

The next matrix represents the position of 11 trees in a property. A cell with a number indicates that the Tree whose identifier is the number in the cell is located in the X,Y position of the garden. For example, tree 11 is in position (0 , 3). In this example, The number of fence posts required will be 6 and the sequence of tree identifiers that will be part of the fence posts is 11,6,4,10,9 and 8.

	0	1	2	3	4
0				11	
1	8		7		
2			1		
3	9	2			6
4				5	
5			3	4	
6	10				

NOTES/HINT

The first tree of the output sequence is always the last tree for which a position was provided in the input file. Your sequence in the output file must be printed in clockwise order.

SAMPLE INPUT FILE

```
11
1 2 2
2 1 3
3 2 5
4 3 5
5 3 4
6 4 3
7 2 1
8 0 1
9 0 3
10 0 6
11 3 0
```

SAMPLE OUTPUT FILE

```
6
11 6 4 10 9 8
```

SAMPLE INPUT FILE

```
9
1 7.1 1.1
2 2.1 2.1
3 9.1 -1.1
4 9.1 2.1
5 10.1 -1.1
6 7.1 0.0
7 5.1 2.1
8 4.1 -2.1
9 4.1 3.1
```

SAMPLE OUTPUT FILE

```
5
9 2 8 5 4
```

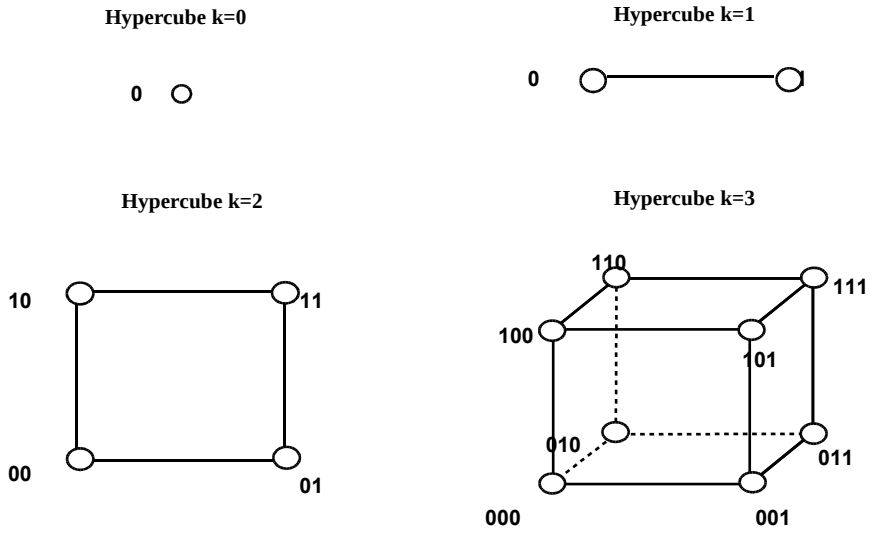
Problem No. 5 – Hypercube Routing
Executable Program : PROG5.EXE, PROG5.CLASS
Source Program: PROG5.CPP, PROG5.JAVA, PROG5.C, PROG5.PAS
Input file: PROG5.IN Output file: PROG5.OUT

A key component of Parallel Computer architecture is the Interconnection Network. This network enables the communication among different processors of the computer. Throughout the years, different interconnection networks have been used. One of the most popular and efficient networks is the hypercube network. This network has been used in different systems such as nCube, Intel IPSC, Thinking Machines CM-2 and the SGI-Gray Origin 2000 supercomputer.

A hypercube network of dimension *k* has the following properties

- The total number of nodes is 2^k.
- The nodes are identified with a node number in the range $0..2^k - 1$
- Every node has *k* bi-directional links with other nodes.
- Two nodes are connected if and only if the binary representation of the node number of the two nodes differs in exactly 1 bit.
- The binary representation of nodes in a hypercube of dimension k has k bits.(the exception is the singular case of hypercube with k=0).

The following figure shows a graphical representation of hypercube of dimensions 0, 1, 2 and 3.



A shortest path between any two nodes of a hypercube is represented as a sequence of nodes No, $N_1 \ldots N_J$, where *J* is the length of the path. In a hypercube, there might be more than one shortest

paths between any two nodes. We are interested in computing a *bit-position ordered shortest path*. A bit-position ordered shortest path is a hypercube shortest path with the following property:

For any pair of nodes N_i, N_{i+1} that belong to the sequence that represents a shortest path, if the difference in the bit representation of nodes N_i and N_{i+1} is in bit position *q* (bit position is considered from right to left) then, the bit position of the difference in the bt representation between any pair of nodes that are in the sequence before Ni should be less than q. Also the bit position of the difference in the bit representation between any pair of nodes that are after N_{i+1} in the sequence should be greater than *q*.
For example, for hypercube with k=3, a *bit-position ordered shortest path* between nodes 1 (001) and 6 (110) is
1 (001),0(000),2 (010) and 6 (110).

A shortest path that *is not bit-position ordered* is 1 (001), 3 (011), 7 (111) and 6 (110). The problem here is that nodes 1 and 3 differ in bit position 2 however, nodes 7 and 6 differ in bit position 1

Write a program that computes the *bit-position ordered shortest path* between any two nodes of the hypercube and also outputs the length of the path.

INPUT

Your program will repeatedly accept the input sets from the input file. Each input set has the dimensionality of a hypercube (the dimension of the hypercube is in the range of 0 up to 20) a source and a destination node (integer numbers in decimal base). The program will stop when it identifies an input data set with dimension 0.

OUTPUT

The program will output the following information:

- The length (number of links that must be traversed) of the shortest path between the source and the destination nodes.
- An ordered shortest path showing all the nodes that are visited from the source to the destination. The nodes must be shown in binary and decimal representation.

SAMPLE INPUT FILE

```
2 0 3
5 20 6
3 2 5
0
```

SAMPLE OUTPUT FILE

```
Hypercube Dimension 2
Source Node 0
Destination Node 3
Shortest Path Length 2
Shortest Path
00 (0)
01 (1)
11 (3)

Hypercube Dimension 5
```

```
Source Node 20
Destination Node 6
Shortest Path Length 2
Shortest Path
10100 (20)
10110 (22)
00110 (6)

Hypercube Dimension 3
Source Node 2
Destination Node 5
Shortest Path Length 3
Shortest Path
010 (2)
011 (3)
001 (1)
101 (5)
```

Problem No. 6 – BST in a matrix
Executable Program : PROG6.EXE, PROG6.CLASS
Source Program: PROG6.CPP, PROG6.JAVA, PROG6.C, PROG6.PAS
Input file: PROG6.IN Output file: PROG6.OUT

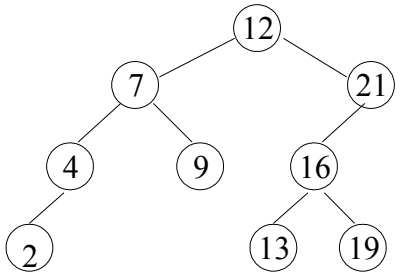
BACKGROUND

The structure (or shape) of a Binary Search Tree (BST) can be represented in memory as follows:

Let d_1, d_2, d_3, … d_n, be the keys (nodes) of a BST where $d_1 < d_2 < d_3 < \ldots d_n$. You can see that each key has an index that identify the key. For example, the key d_3 has the index 3.

The matrix M is built as follows: for every i <=j, M[i,j] contains the index of the key that represents the root in the BST that is formed with the keys from d_i to d_j. The rest of the values in the matrix M are set to zero since are nor required. This representation is known as Matrix of Roots.
The Matrix of Roots has n+1 rows that range from 1 to n+1 and n+1 columns that range from 0 to n.

Below is an example of a BST and its corresponding Matrix of Roots:



	0	1	2	3	4	5	6	7	8	9
1	0	1	2	3	3	5	5	5	5	5
2	0	0	2	3	3	5	5	5	5	5
3	0	0	0	3	3	5	5	5	5	5
4	0	0	0	0	4	5	5	5	5	5
5	0	0	0	0	0	5	5	5	5	5
6	0	0	0	0	0	0	6	7	7	9
7	0	0	0	0	0	0	0	7	7	9
8	0	0	0	0	0	0	0	0	8	9
9	0	0	0	0	0	0	0	0	0	9
10	0	0	0	0	0	0	0	0	0	0

INPUT

You will be given a number of test cases. Each test case represents a BST in its Matrix of Roots representation. The input for each test is as follows: The first line represents the order (# of rows and columns) of the Matrix of Roots for the BST. The following lines will represent the values of the matrix, each line will correspond to one row of the matrix. The values within each row are delimited by tabs. All values are non-negative and you can assume that a BST contains at most 100 elements.
You will identify the end of input when you find a single line with a -1 value.

OUTPUT

For each test case, print the case number followed by a messages indicating the height of the BST and the average number of comparisons that will be required to perform the search of keys in the BST. The average number of comparisons is the sum of the comparisons that each key needs to be found divided into the total quantity of keys. You are going to count only the comparisons

between keys. For example, in the BST of above, the key 12 needs 1 comparison to be found; the key 19 needs 4 comparisons to be found, etc.

Output the information using the following format:

```
Case X:
The height of the BST is: Y
The average of the search in the BST is: Z
```

where `X`, `Y` and `Z` are replaced by the appropriate values. Z could be displayed with 2 decimals.

SAMPLE INPUT

```
10
0     1     2     3     3     5     5     5     5     5
0     0     2     3     3     5     5     5     5     5
0     0     0     3     3     5     5     5     5     5
0     0     0     0     4     5     5     5     5     5
0     0     0     0     0     5     5     5     5     5
0     0     0     0     0     0     6     7     7     9
0     0     0     0     0     0     0     7     7     9
0     0     0     0     0     0     0     0     8     9
0     0     0     0     0     0     0     0     0     9
0     0     0     0     0     0     0     0     0     0
-1
```

SAMPLE OUTPUT

```
Case 1:
The height of the BST is: 4
The average of the search in the BST is: 2.89
```