acm International Collegiate Programming Contest
sponsored by IBM

Problem 1: Numbered Permutations

Executable Program : Prog1.exe, Prog1.class
Source Program: Prog1.cpp, Prog1.java, Prog1.pas
Input: Prog1.in, Output: Standard Output

Consider a set of n different elements that can be permuted in n! different ways. Such permutations can be first, ordered lexicographically and then they can numbered. For example, the set {a, b, c} admits six permutations that can be ordered as follows:

```
1  abc
2  acb
...
6  cba
```

This example shows the 1st, 2nd and 6th permutation, Write a program that can compute the kth permutation of a set of n elements in less than 1 minute. The elements of such a set are that character c and the n -1 ASCII characters that follow it. The program must read a text file where each line is of the form:

```
k  n  c
```

where k and n are integers: 1 <= n <= 20, 1 <= k <= n!,
and c is an ASCII character.

For each line of the file read by the program, give the corresponding permutation.

Input example:	Output example:
1 3 a	abc
2 3 a	acb
6 3 A	CBA
1 4 a	abcd
1 4 C	CDEF
24 4 1	4321
1 20 a	abcdefghijklmnopqrst

acm International Collegiate Programming Contest

sponsored by IBM

Problem 2: Centipede Collisions

Executable Program : Prog2.exe, Prog2.class
Source Program: Prog2.cpp, Prog2.java, Prog2.pas
Input: Prog2.in, Output: Standard Output

A small boy named Tommy has some toy centipedes that are a series of 1 centimeter segments. Tommy assembles his centipedes to any length he likes and places them on a 30x30 centimeter board that allows the centipedes to travel in 1 centimeter wide tracks that criss-cross the board. The centipedes travel only parallel to either the x or y axis on the board. Centipede segments of the same centipede advance at the same time and centipedes advance in cyclic numerical order (all of centipede 0 first, then 1, etc.). When more than one segment of two or more centipedes occupy the same x,y coordinate, there is a centipede collision. Anytime a collision occurs, all segments arriving at the collision site are retained at this point like a black hole. All remaining segments on a centipede detach from the segment involved in the collision and continue their march until another collision occurs or an existing collision site is encountered or until the segments fall off the edge of the board. Anytime a segment enters a collision site, it becomes part of the collision.

Since Tommy left home without his centipede set, his mother has hired you to write a simulation program for his entertainment. Your program will simulate his board with a text printout of his grids. For example, Tommy may simulate 5 centipedes on his board that start out as shown on the grid on the left and finish as shown on the grid on the right (note the example grid is only 10x10 whereas Tommy's is 30x30).

```
9    . . . . . . . . . .          9    . . . . . . . . . .
8    . . . . . . . . . .          8    . . . . . . . . . .
7    1 1 1 1 1 . . . . .          7    . . . . . X . . . X
6    . 0 . . . . . . . .          6    . . . . . . . . . .
5    . 0 . . . . . . . 3          5    . . . . . . . . . .
4    . 0 . . . 2 . . . 3          4    . . . . . . . . . .
3    . 0 . . . 2 . . . 3          3    . . . . . . . . . .
2    . . . . . 2 . . . 3          2    . . . . . . . . . .
1    . . . . . 2 . . . 3          1    . . . . . . . . . .
0    . . . . . 2 4 4 4 3          0    . X . . . . . . . .
Y                                 Y
 / X 0 1 2 3 4 5 6 7 8 9           / X 0 1 2 3 4 5 6 7 8 9
```

```
Where 0 represents a centipede segment traveling from top to bottom,
      1 represents a centipede segment traveling from left to right,
      2 represents a centipede segment traveling from bottom to top,
      3 represents a centipede segment traveling from bottom to top,
      4 represents a centipede segment traveling from right to left, and
      X represents a collision site involving 2 or more segments.
```

Your program will simulate up to 10 centipedes that travel on a 30x30 board. Tommy has 100 segments that he may use in his simulation. Of course, no centipede can be longer than 30 segments.

Input

Input to your program will consist of a series of simulation sets. The first line of each input set will be a single integer (1<=N<=10) representing the number of centipedes in the simulation. (Centipedes are numbered 0 through N-1 in the same order as the input.) The next N lines will each represent one centipede and will contain a single direction character followed by 3 integers. The direction character can be 'U', 'D', 'L', or 'R' for "Up", "Down", "Left", or "Right" respectively and indicate the direction of travel. The next integer (1<=L<=30) indicates the length of the centipede in segments. The second and third integers indicate the x,y coordinates of the lead segment (0<=x&y<=29). The next L-1 segments of the centipede occupy the L-1 board positions extending in the opposite direction from the direction of travel starting adjacent to the lead segment. You can be assured that no centipede segment will originate off the board nor will the original configuration involve any collisions.

Output

For each input simulation set, you should print (exactly) the following lines as the first two lines of output (starting in column 4).

```
123456789 …         <--         (Columns)

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

The next 30 lines represent the state of the board after the simulation has completed (all centipede segments have entered a collision site or fallen off the edge of the board). Columns 1 and 2 should contain the row number (rows are numbered 29 down to 00) with leading zeroes. Even numbered columns between 4 and 62 (inclusive) represent the contents of the board cells. Board cells can contain either an X or a period. A period represents an empty cell and X represents a cell that contains 2 or more centipede segments involved in a collision. The last line of each output set is a blank line.

Follow the Sample Output for the exact format of the expected output.

Sample Input

```
10
R 9 11 23
U 8 11 17
U 5 15 15
U 5 15 8
D 9 23 13
U 6 23 6
R 9 8 9
L 13 17 0
U 12 13 11
L 5 20 9
```

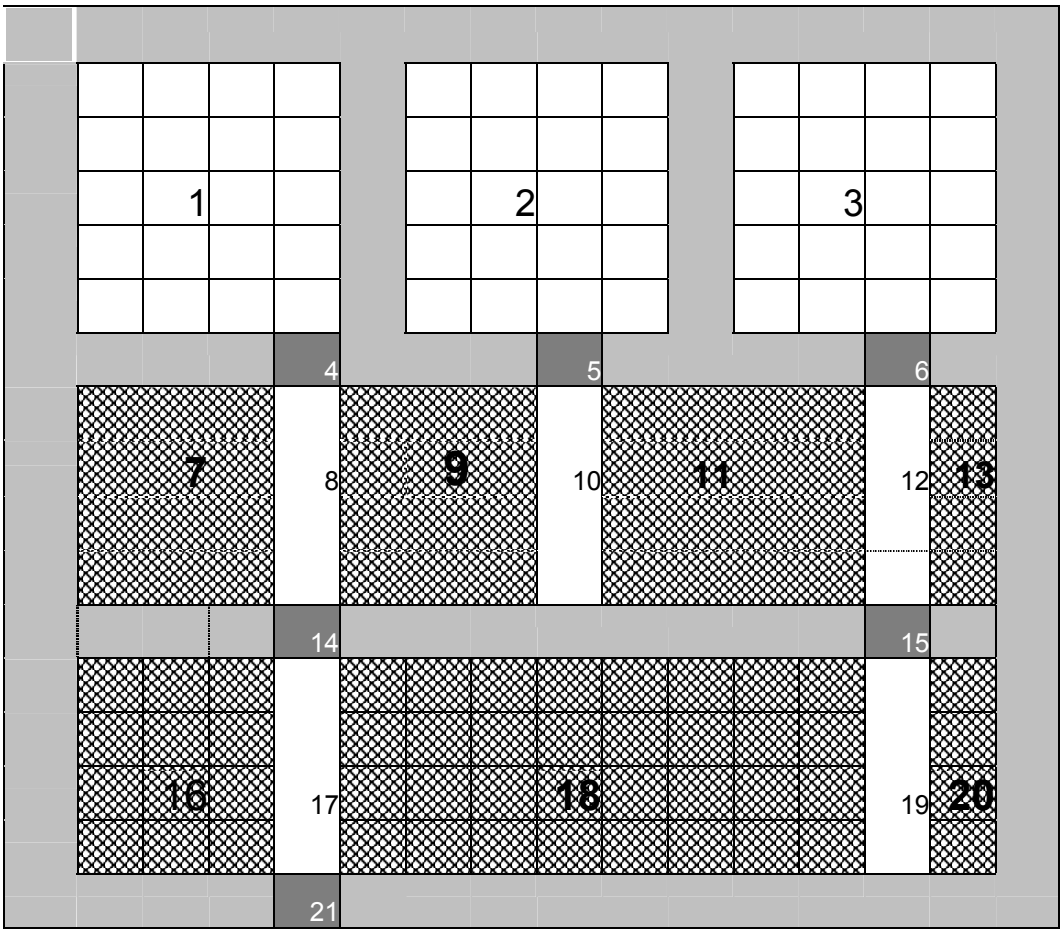
Sample Output

```
   00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
29 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
28 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
27 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
26 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
25 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
24 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
23 .  .  .  .  .  .  .  .  .  .  .  X  .  .  .  X  .  .  .  .  .  .  .  .  .  .  .  .  .  .
22 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
21 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
20 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
19 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
18 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
17 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
16 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
15 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
14 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
13 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
12 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
11 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
10 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
09 .  .  .  .  .  .  .  .  .  .  .  .  X  .  X  .  .  .  .  .  .  .  X  .  .  .  .  .  .  .
08 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
07 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
06 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
05 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
04 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
03 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
02 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
01 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
00 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
```

acm International Collegiate Programming Contest
sponsored by IBM

Problem No. 3 Mobil Cyclop Robot
Executable Program : Prog3.exe, Prog3.class
Source Program: Prog3.cpp, Prog3.java, Prog3.pas
Input: Prog3.in, Output: Standard Output

You have a programmable mobil cyclop robot with a camera mounted on itself . It has the ability to identify different areas in a building, using the numbers assigned to each one of them in a four-neighborhood (by example, if the robot is in area 8, it can visualize his neighbors areas 7, 4, 9 and 14).
You have to design a program for the robot, in such a way that the robot can find the shortest length path, between any two data points without visiting twice any area. The length of the path is calculated with the number of visited areas, including the starting and the final areas.
In case of no possible path, the program should show the message: "*There is not possible path*"
Taking into account that the maximum number of areas is 99 and the minimum 2.



INPUT

The first line of the input file contains an integer *n* representing the number of areas. The next *n* lines will have positive integers. The first one representing the area and the others, the neighborhoods. The last line will have two integers representing the start and final areas.

OUTPUT

An integer representing the length of the shortest path or the string *"There is not possible path*"

SAMPLE INPUT FILE

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

SAMPLE OUTPUT FILE

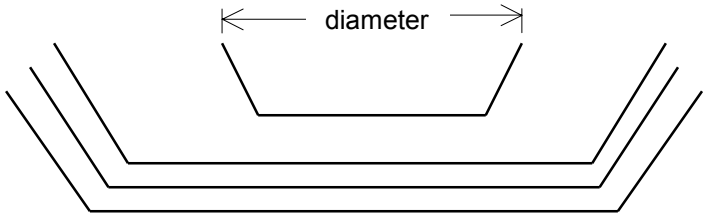
```
8
```

acm International Collegiate Programming Contest
sponsored by IBM

Problem No. 4 Baskets

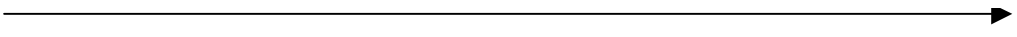
Executable Program : Prog4.exe, Prog4.class
Source Program: Prog4.cpp, Prog4.java, Prog4.pas
Input: Prog4.in, Output: Standard Output

Juan lives in the mountains. He waits all the year for the town's party, because tradesmen from all over the country will come to show and sell their merchandise like baskets, mud pots and a lot of craftsmanship. This year he is planning to buy some baskets. All the baskets are round, having diameter, thickness and colors that make them different from each other. Nevertheless, the problem is how are they going to be carried, because the only truck he owns is small. He also has time restrictions because the stands do not have their own parking space, so he is planning to park in front of the chosen stand, buy the selected basket (only one) and quickly put it in the truck.
To do this, Juan has decided to cross the stand's street walking with the idea of looking for the merchandise and choose the baskets he would like to buy. He had planned to measure the diameters of all the baskets he would like to buy and write them in his notebook. In this way when he had walked all the stand's street, he would have gotten all the diameter information to decide where he is going to stop his truck to buy. Each basket he buys must keep inside the previous one bought making a stack like is shown in the figure. Suppose the basket's thickness despicable and no baskets rearrangements can be made because the time restriction.

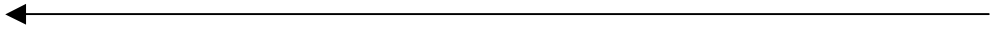


In his walk by the stand's street, Juan had picked up the following diameters of baskets he would like to buy. The diameters are :

Walking Path through the stands selecting baskets and picking up the data



200	1300	1800	1000	100	2000	2500	2900	1100	3000	1200	2400



Truck Path through the stands buying pieces

To this diameters list Juan has realized that the maximum amount of baskets he can buy is 7.

Input contains multiple cases. Each test case consists of a line containing a single integer *n*, (for n < 10000) representing the amount of baskets in the list. In the next *n* lines, integers representing the diameters of each one of the *n* baskets. A line containing a –1 follows the last test case.

Sample Input

```
12
200
1300
1800
1000
100
2000
2500
2900
1100
3000
1200
2400
5
200
300
500
100
3000
4
1000
700
500
200
-1
```

Output for Sample Input

```
7
4
1
```

acm International Collegiate Programming Contest
sponsored by IBM

Problem 5: STM

Executable Program : Prog5.exe, Prog5.class
Source Program: Prog5.cpp, Prog5.java, Prog5.pas
Input: Prog5.in, Output: Standard Output

Efficient Developments ships a product which requires some database features. Since the company can't assume an existing DBM installed in the client system and their distribution must be as small as possible. Their solution is to create a Simple Transaction Manager.

This Simple Transaction Manager must be able to retain information through its executions and to avoid several client processes to access the same data at the same time. Client process will be identified, on the test file, as a three uppercase letter string followed by a colon, the command the client is requesting with its corresponding parameters, if any. The commands that can be issue to the STM are:

STARTSERVER
SHUTDOWN
CRASH
STARTTRANSACTION
COMMIT
ROLLBACK
READ Record
UPDATE Record Value
CREATEDATABASE Size

The database has at most 500 registers but it must support to read a four digits size number (such as 0500) in what can be thought of as a single table. All values and all record keys have the four digits, left zero padded numbers are used.

In order to access the server, it must be running. The server will support only one transaction per client at the same time, and all records accessed (read or write) while the transaction is open will be locked for the other clients. When an operation which requires a running server is performed against an stopped server (all except crash, createdatabase and startserver) the response: "ERROR: NO SERVER UP" is generated,

The database can only be created when the server is not running,
The server cannot be started while running, nor shutted down if it not running
The database must be started before it can be shutted down.
To shutdown a database there must be no pending transactions.

The server can be “crashed” at any time: it is stopped, regardless of its status or pending transactions.

When a transaction is committed, changes are made permanent. The information should exist between calls to the STM. A CREATEDATABASE statement will destroy all existing data and initialize all records to “0000”. A record update outside a transaction is permanent. When a Transaction is rolledback, all updates to records within that transaction are discarded.

The STM outputs to a file, the client who issued a command, and the command output:

Please see sample output files.

The following table shows the possible message the STM systems should produce.

Command	*Success*	*Error Message*	*Error Message*
STARTSERVER	OK	ERROR: SERVER UP	
SHUTDOWN	OK	ERROR: NO SERVER UP	ERROR: OPEN TRANSACTIONS
CRASH	OK		
STARTTRANSACTION	OK	ERROR: A TRANSACTION IS ALREADY OPEN	
COMMIT	OK	ERROR: NO OPEN TRANSACTION	
ROLLBACK	OK		
READ Record	Value	ERROR: RECORD IS LOCKED	
UPDATE Record Value	OK	ERROR: RECORD IS LOCKED	
CREATEDATABASE Size	Value	ERROR: SERVER UP	

NO Read/Update operation will try to access non existing records at database.
The server is shutted, not shouted!!! (the word is misspelled all along)

SAMPLES

Input:

```
YYY: CREATEDATABASE 500
XXX: STARTSERVER
AAA: READ 0001
AAA: UPDATE 0001 1222
AAA: STARTTRANSACTION
AAA: UPDATE 0001 1500
BBB: STARTTRANSACTION
AAA: READ 0001
BBB: UPDATE 0315 0123
CCC: STARTTRANSACTION
DDD: UPDATE 0316 0456
CCC: STARTTRANSACTION
CCC: READ 0001
BBB: COMMIT
XXX: SHUTDOWN
CCC: UPDATE 0317 0789
CCC: COMMIT
XXX: CRASH
AAA: UPDATE 0001 1300
DDD: READ 0245
```

Output:

```
YYY: OK
XXX: OK
AAA: 0000
AAA: OK
AAA: OK
AAA: OK
BBB: OK
AAA: 1500
BBB: OK
CCC: OK
DDD: OK
CCC: ERROR: A TRANSACTION IS ALREADY OPEN
CCC: ERROR: RECORD IS LOCKED
BBB: OK
XXX: ERROR: OPEN TRANSACTIONS
CCC: OK
CCC: OK
XXX: OK
AAA: ERROR: NO SERVER UP
DDD: ERROR: NO SERVER UP
```

Input:

```
AAA: READ 0001
BBB: READ 0315
DDD: READ 0316
CCC: READ 0317
DDD: READ 0245
```

Output:

```
AAA: ERROR: NO SERVER UP
BBB: ERROR: NO SERVER UP
DDD: ERROR: NO SERVER UP
CCC: ERROR: NO SERVER UP
DDD: ERROR: NO SERVER UP
```

Input:

```
XXX: STARTSERVER
AAA: READ 0001
BBB: READ 0315
DDD: READ 0316
CCC: READ 0317
DDD: READ 0245
```

Output:

```
XXX: OK
AAA: 1222
BBB: 0123
DDD: 0456
CCC: 0789
DDD: 0000
```