

CTU Open Contest 2003

Storehouse

`store.p`, `store.c`, `store.C`

Advanced Cargo Movement, Ltd. owns a big storehouse in which a lot of various goods is stored. The storehouse has limited number of bays where a cargo can be loaded. Each day, trucks come to the bays, load their cargo (each truck loads just one type of goods) and leave for the shops. In order to speedup the loading, storekeepers move the goods to the bay in advance. Since the exact quantity of cargo to be loaded is not known beforehand, the storekeepers must prepare more goods than needed and the left-over goods is then returned to the store. Thus, it is useful when the next truck coming to the bay loads the same goods as the previous one, because it saves unnecessary movement of cargo. The capacity of trucks is much smaller than the capacity of the bay, therefore, any number of trucks can be served from one bay without reloading, as long as they want to load the same type of goods.

Your task is to organize which type of goods will be prepared in which bay, so that storekeepers must move the unloaded goods back to the storehouse as few times as possible. At the beginning of each day, no goods is prepared in any bay. The goods left in bays at the end of the day is not counted to the number of moves needed.

Input Specification

The first line of the input contains the number of test cases to solve. Each test case starts with the line containing three integer numbers, B, G, N, $1 \leq B \leq 1\,000$, $1 \leq G \leq 1\,000\,000$, $1 \leq N \leq 1\,000\,000$, separated by a single space. B is the number of bays in the storehouse, G the number of types of goods stored in the storehouse and N the number of trucks coming to the storehouse. The test case continues by N lines. On the i-th line, there is a number t_i, $1 \leq t_i \leq G$, specifying the type of goods the i-th truck wants to load.

Output Specification

For each test case, your program should output "`Case` X`:`" on the first line, where X is the case number starting with one. Then N lines follow. For each i, $1 \leq i \leq N$, the i-th line must contain one of the following strings:

- "`NO ACTION`" — when no action needs to be performed before the arrival of the truck i (i.e., the goods i-th truck wants to load is already at some bay), or
- "`LOAD` b g" — when the goods with number g should be moved to the bay b before the i-th truck arrives. The goods currently present at the bay b is automatically moved back to the storehouse.

Each time some truck arrives, the goods it wants to load must be prepared at some bay. Assume that any truck leaves the bay before the next one arrives, i.e., only a single truck is served at a time.

The number of "`LOAD`" actions should be as small as possible. If there are more solutions with the same number of "`LOAD`" actions, choose any one you want. Outputs for different test cases should be separated by a blank line.

Sample Input

```
2
2 4 5
1
2
1
4
1
3 3 3
1
3
2
```

Output for Sample Input

```
Case 1:
LOAD 1 1
LOAD 2 2
NO ACTION
LOAD 2 4
NO ACTION

Case 2:
LOAD 1 1
LOAD 2 3
LOAD 3 2
```

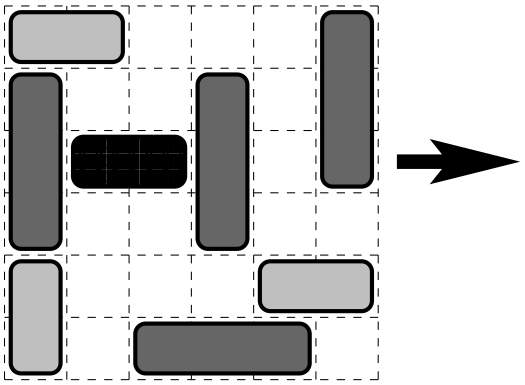
Czech ACM Student Chapter
CTU in Prague, Dept. of Computer Science and Engineering
Charles University in Prague, MFF
VŠB TU Ostrava, FEI, Dept. of Computer Science
SUT Bratislava, FEI, Dept. of Computer Science and Engineering

Traffic Jam

`traffic.p`, `traffic.c`, `traffic.C`

Traffic jam is a real nightmare of all drivers. Nobody likes to be stuck in the overfilled streets, when the cars move very slowly, if they even move at all. Professional drivers face traffic jams quite often. Truck drivers of ACM are not an exception. Can you help drivers to find the way out of the traffic jam?

We can model a small (but complicated) traffic jam on a 6×6 grid of squares. Vehicles (cars and trucks) are scattered over the grid at integer locations, as shown below. Both types of vehicles are 1 square wide. Cars are 2 squares long, and trucks are 3 squares long. Vehicles may be oriented either horizontally (East-West) or vertically (North-South) relative to the grid.



Vehicles cannot move through each other, cannot turn, and cannot move over the edge of the grid. They can move in their direction (horizontally-oriented vehicles cannot move vertically and vice versa), as long as they are not blocked by another vehicle or by the edge of the grid. Only one vehicle may move in a single step, but it may move by as many squares at a time as possible, providing there is enough empty space.

Our goal is to move vehicles back and forth until a particular horizontally-oriented vehicle (your own car — the black one on the picture above) leaves the rightmost (eastern-most) edge of the grid, where it is considered to have escaped the traffic jam. You are to write a program that will find a solution requiring the minimum possible number of moves.

Input Specification

The input file will consist of one or more input scenarios. Each scenario begins with a single integer n, $1 \leq n \leq 10$, giving the number of vehicles in the scenario. Then there will be 6 lines of input, each containing 6 characters. Each character is either a dot ("`.`") representing an empty square, or a lowercase letter representing a vehicle. Your own vehicle is always oriented horizontally and represented by "`x`" characters. The other vehicles use the letters sequentially, beginning with "`a`".

The last scenario will be followed by a line containing a single zero.

Output Specification

For each scenario, output a single line with the statement "`Scenario #`K `requires` X `moves.`", where K is the number of the scenario (starting with 1) and X is the minimum number of moves required to escape the the traffic jam with the particular car.

If it is not possible to escape, output the sentence "`You are trapped in scenario #`K`.`" instead.

Sample Input

```
8
aa...b
c..d.b
cxxd.b
c..d..
e...ff
e.ggg.
8
abbc..
a..c..
axxc..
..gddd
..g..e
..fffe
0
```

Output for Sample Input

```
Scenario #1 requires 8 moves.
Scenario #2 requires 25 moves.
```