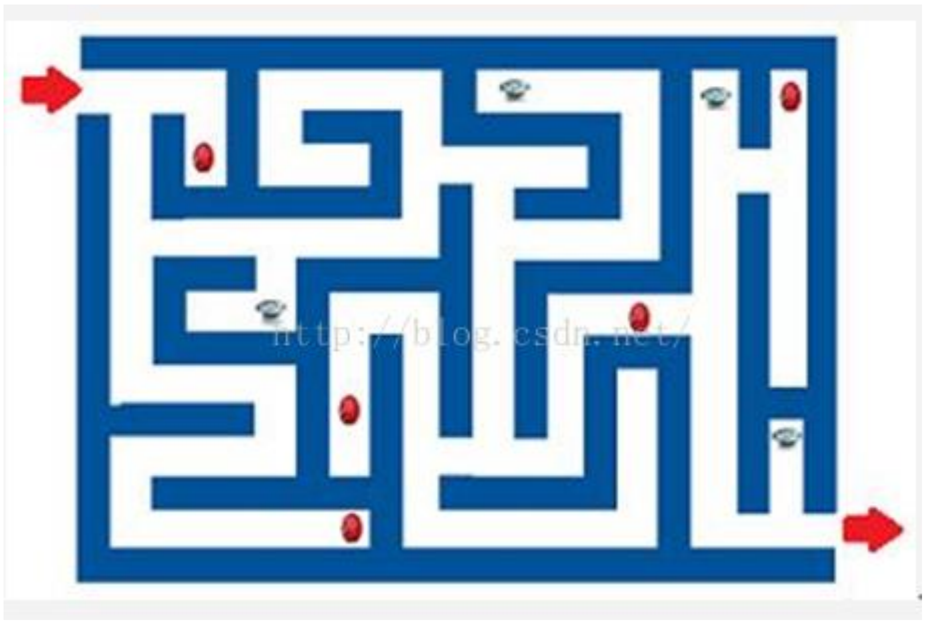
Picking up Jewels



Picking up Jewels There is a maze that has one entrance and one exit. Jewels are placed in passages of the maze. You want to pick up the jewels after getting into the maze through the entrance and before getting out of it through the exit. You want to get as many jewels as possible, but you don't want to take the same passage you used once.

When locations of a maze and jewels are given, find out the greatest number of jewels you can get without taking the same passage twice, and the path taken in this case.

Time limit: 1 sec (Java: 2 sec) (If your program exceeds this time limit, the answers that have been already printed are ignored and the score becomes it may be better to print a wrong answer when a specific test case might cause your program to exceed the time limit. One guide for the time limit excess be the size of the input.)

[Input]

There can be more than one test case in the input file. The first line has T, the number of test cases. Then the totally T test cases are provided in the following lines (T ≤ 10).

In each test case, In the first line, the size of the maze N (1 ≤ N ≤ 10) is given. The maze is N×N square-shaped. From the second line through N lines, information of the maze is given. "0" means a passage, "1" means a wall, and "2" means a location of a jewel. The entrance is located on the upper-most left passage and the exit is located on the lower-most right passage. There is no case where the path from the entrance to the exit doesn't exist.

[Output]

For each test case, you should print "Case #T" in the first line where T means the case number. For each test case, from the first line through N lines, mark the path with 3 and output it. In N+1 line, output the greatest number of jewels that can be picked up.

[I/O Example]

Input

```
2

5

0 0 0 2 0

2 1 0 1 2

0 0 2 2 0

0 1 0 1 2

2 0 0 0 0

6

0 1 2 1 0 0

0 1 0 0 0 1

0 1 2 1 2 1

0 2 0 1 0 2

0 1 0 1 0 1

2 0 2 1 0 0
```

Output

```
Case #1

3 0 3 3 3
3 1 3 1 3
3 0 3 2 3
3 1 3 1 3
3 3 3 0 3
6

Case #2

3 1 2 1 0 0
3 1 3 3 3 1
3 1 3 1 3 1
3 2 3 1 3 2
3 1 3 1 3 1
3 3 3 1 3 3
4
```