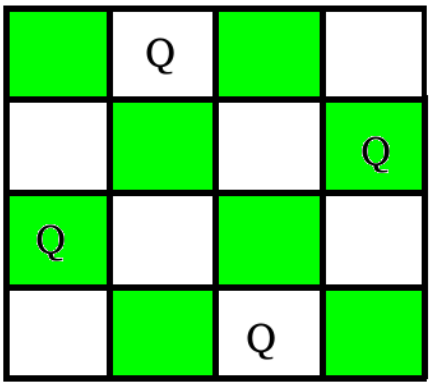
N-Queen Problem

The n-queens puzzle is the problem of placing n queens on an n×n chessboard such that no two queens attack each other. Given an integer n, print all distinct solutions to the n-queens puzzle. Each solution contains distinct board configurations of the n-queens' placement, where the solutions are a permutation of [1,2,3..n] in increasing order, here the number in the *ith* place denotes that the *ith*-column queen is placed in the row with that number. For eg below figure represents a chessboard [3 1 4 2].



Input:
The first line of input contains an integer **T** denoting the no of test cases. Then T test cases follow. Each test case contains an integer n denoting the size of the chessboard.

Output:
For each test case, output your solutions on one line where each solution is enclosed in square brackets '[', ']' separated by a space . The solutions are permutations of {1, 2, 3 …, *n*} in increasing order where the number in the ith place denotes the ith-column queen is placed in the row with that number, if no solution exists print -1.

Constraints:
1 <= T <= 10
1 <= n <= 10

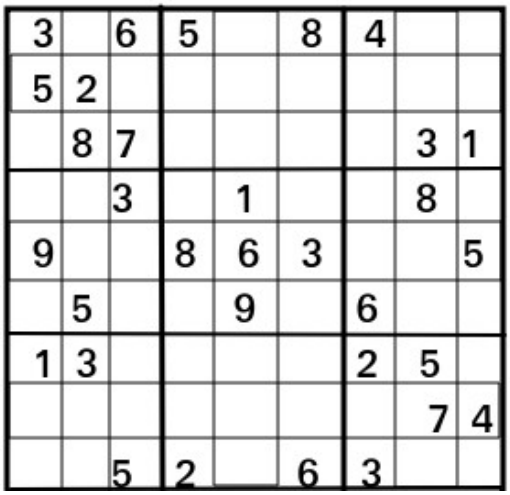
Example:
Input
2
1
4

Output:
[1]
[2 4 1 3] [3 1 4 2]

Solve the Sudoku

Given an incomplete Sudoku configuration in terms of a 9 x 9 2-D square matrix (mat[][]). The task to print a solved Sudoku. For simplicity you may assume that there will be only one unique solution.

Sample Sudoku for you to get the logic for its solution:



Input:
The first line of input contains an integer T denoting the no of test cases. Then T test cases follow. Each test case contains 9*9 space separated values of the matrix mat[][] representing an incomplete Sudoku state where a 0 represents empty block.

Output:
For each test case, in a new line, print the **space separated values** of the solution of the the sudoku.

Constraints:
1 <= T <= 10
0 <= mat[] <= 9

Example:

Input:

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

Output:

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

Explanation:

Testcase 1: The solved sudoku is:

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

Rat in a Maze Problem - I

Consider a rat placed at **(0, 0)** in a square **matrix of order N*N**. It has to reach the destination at **(N-1, N-1)**. Find all possible paths that the rat can take to reach from source to destination. The directions in which the rat can move are 'U'(up), 'D'(down), 'L' (left), 'R' (right). Value 0 at a cell in the matrix represents that it is blocked and cannot be crossed while value 1 at a cell in the matrix represents that it can be traveled through.

Example 1:

```
Input: N = 4, m[][] = {{1, 0, 0, 0},
                       {1, 1, 0, 1},
                       {1, 1, 0, 0},
                       {0, 1, 1, 1}}
Output: DDRDRR DRDDRR
Explanation: The rat can reach the
destination at (3, 3) from (0, 0) by two
paths ie DRDDRR and DDRDRR when printed
in sorted order we get DDRDRR DRDDRR.
```

Example 2:

```
Input: N = 2, m[][] = {{1, 0},
                       {1, 0}}
Output: -1
Explanation: No path exits
```

Your Task:
You don't need to read input or print anything. Complete the function **printPath()** which takes **N** and 2d array **m[][]** as input parameters and returns a list of paths.

Note: In case of no path, return an empty list. The driver will output **-1** automatically.

Expected Time Complexity: $O((N^2)^4)$.
Expected Auxiliary Space: O(L*X), L = length of the path, X = number of paths.

Constraints:
$2 \leq N \leq 5$
$0 \leq m[i][j] \leq 1$

Generate IP Addresses

Given a string **S** containing only digits, Your task is to complete the function **genIp()** which returns a vector containing all possible combination of valid IPv4 ip address and takes only a string **S** as its only argument.
Note : Order doesn't matter.

For string 11211 the ip address possible are
1.1.2.11
1.1.21.1
1.12.1.1
11.2.1.1

Example 1:

```
Input:
S = 1111
Output: 1.1.1.1
```

Your Task:

Your task is to complete the function **genIp()** which returns a vector containing all possible combination of valid IPv4 ip address in sorted order and takes only a string **S** as its only argument.

Expected Time Complexity: O(N * N * N * N)
Expected Auxiliary Space: O(N * N * N * N)

Constraints:
1<=N<=16
here, N = length of S.
S only contains digits(i.e. 0-9)
