**Grading and Policies**

1. Number of team members is 5, not more than 6

2. There are 10 ideas, each one will be implemented by 5 teams.

**Deliverables**

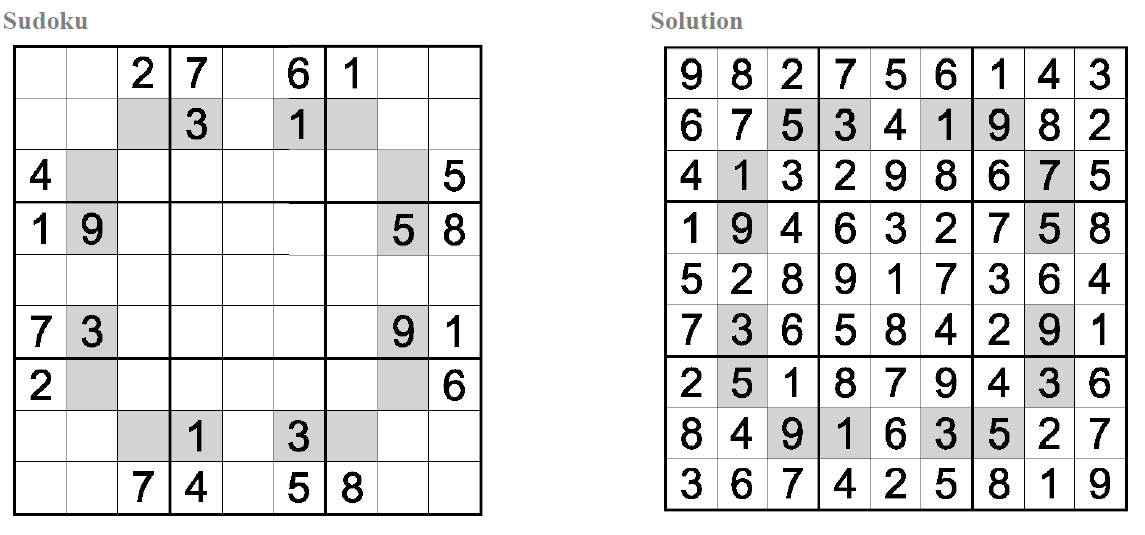
1. CD contains the runnable code

2. Printed documentation includes pseudo-code of algorithm, Sequential and Parallel version Code

**Projects Ideas**

1. ***Sudoku***

Place a digit from 1 to 9 into each of the empty squares so that each digit appears exactly once in each of the rows, columns and the nine outlined 3x3 regions. The grey squares must contain odd digits.



Now, ahmed faced with a problem, want you to give him to solve. To give you 9 \* 9 grid, in this grid has been filled in part of the yard numbers. Now is this game will have any solution? No solution output of NO, or output YES.

**Input**

The first line cotains the integer *N* (1 ≤ *N* ≤ 100) representing the number of test cases.

Each test case contains 10 lines:

First 9 lines: each line contains 9 characters. The meanings of valid characters: -- '1' - '9' : the number of this grid. -- '?' : Blank Grid.

The last line is a blank line.

**Output**

For each test case, you are to determine whether or not it has solution. Print a line containing "YES" if it has, and "NO" if not.

1. ***Maze***

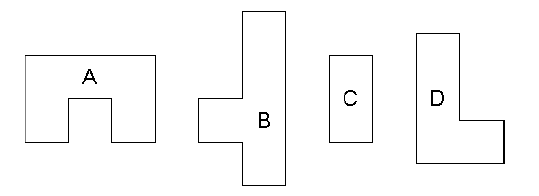
You’ve landed on an alien planet, in the middle of some strange maze, and you want to find the quickest way out.

The maze can be thought of as a m×n grid where some of the squares in the grid are blocked. You know the entire maze (including the size of the grid, which squares are blocked, and your current location). You can drive north, south, west, or east from each square in your hovercar, but not diagonally. Unfortunately, your hovercar was damaged during the rough landing onto the planet, and it can’t turn left or make U-turns: the hovercar can only go straight or turn right. For instance, if you leave the current square by going north, you’ll only be able to leave the next square by going north or east; west is not possible, because that would require a left turn, and south is not possible either, because would require a U-turn. Your hovercar was landed facing north, so on your first move you will need to drive north.

Find an efficient algorithm to find the shortest route to escape from the maze and implement it .

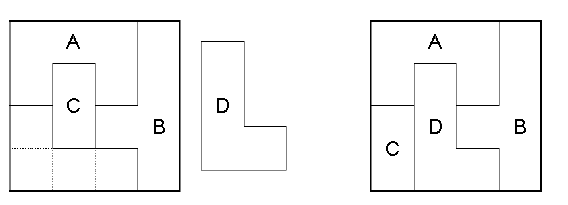
1. ***Make a square***

Make a square with size 5X5 by using 4 or 5 pieces. The pieces can be rotated or flipped and all pieces should be used to form a square. Example set of pieces.



There may be more than one possible solution for a set of pieces, and not every arrangement will work even with a set for which a solution can be found. Examples using the above set of pieces…

Rotate piece D 90 degree then flip horizontal {R 90 + F H}



**Input:**

The first line contains number of pieces. Each piece is then specified by listing a single line with two integers, the number of rows and columns in the piece, followed by one or more lines which specify the shape of the piece. The shape specification consists of 0 or 1characters, with the 1 character indicating the solid shape of the puzzle (the 0 characters are merely placeholders). For example, piece A above would be specified as follows:

2 3

111

101

**Output:**

Your program should report all solution, in the format shown by the examples below. A 4-row by 4-column square should be created, with each piece occupying its location in the solution. The solid portions of piece #1 should be replaced with `1' characters, of piece #2 with `2' characters. Last line displays the pieces which are changed in the original form. If the piece rotate with angle 90{R 90}, and flip vertical or horizontal {F V or F H}. For cases which have no possible solution simply report "No solution possible".

1. ***Crossword Puzzle Problem***

http://www.cs.columbia.edu/~evs/ais/finalprojs/steinthal/

1. ***Examination Timetabling Problem***

http://www.mdpi.com/1999-4893/7/3/295/pdf

1. ***Cyclic redundancy check Algorithm***

https://en.wikipedia.org/wiki/Cyclic\_redundancy\_check

1. ***Image Compression using Huffman Coding***  
   http://courses.cs.washington.edu/courses/cse373/02wi/slides/ImageADT/sld012.htm
2. ***Boundary Extraction and Region Filling using Morphological Image Processing***  
   (chapter 9 from Digital Image Processing by Gonzalez & Woods 2th edition)
3. ***find pattern in the Image using hit and miss transformation***  
   (chapter 9 from Digital Image Processing by Gonzalez & Woods 2th edition)
4. ***implement Histogram matching***  
   (chapter 3 from Digital Image Processing by Gonzalez & Woods 2th edition)