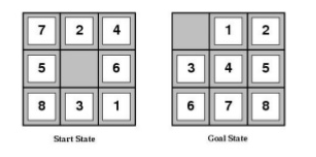
**LAB 8**

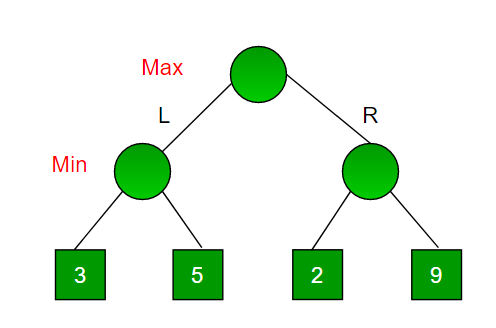
**PRELAB :**

1. Define Cost Function and idea cost function in 8- puzzle problem.

Take the below picture as an example and calculate the heuristic functions for it (using the most two commonly used techniques).



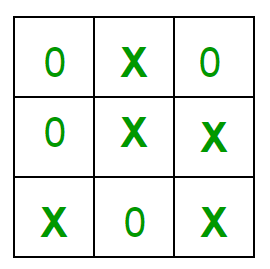
1. Consider a game which has 4 final states and paths to reach the final state are from root to 4 leaves of a perfect binary tree as shown below. Assume you are the maximizing player and you get the first chance to move, i.e., you are at the root and your opponent at the next level. Which move you would make as a maximizing player considering that your opponent also plays optimally?



**INLAB:**

1. Implementation of Tic-Tac-Toe game

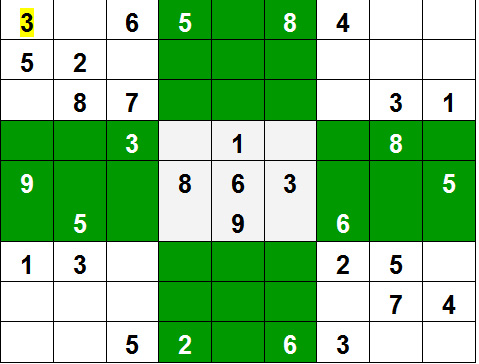
Rules of the Game

* The game is to be played between two people (in this program between HUMAN and COMPUTER).
* One of the players chooses ‘O’ and the other ‘X’ to mark their respective cells.
* The game starts with one of the players and the game ends when one of the players has one whole row/ column/ diagonal filled with his/her respective character (‘O’ or ‘X’).
* If no one wins, then the game is said to be drawn.  
    
  Implementation  
  In our program the moves taken by the computer and the human are chosen randomly. We use the rand() function for this.  
  What more can be done in the program?  
  The program is not played optimally by both sides because the moves are chosen randomly. The program can be easily modified so that both players play optimally (which will fall under the category of Artificial Intelligence). Also the program can be modified such that the user himself gives the input (using scanf() or cin).  
  The above changes are left as an exercise to the readers.  
  Winning Strategy – An Interesting Fact  
  If both the players play optimally then it is destined that you will never lose (“although the match can still be drawn”). It doesn’t matter whether you play first or second.In other ways – “ Two expert players will always draw ”.  
  Isn’t this interesting ?

**POST- LAB :**

1.Sudoku | Backtracking-7

Given a partially filled 9×9 2D array ‘grid[9][9]’, the goal is to assign digits (from 1 to 9) to the empty cells so that every row, column, and subgrid of size 3×3 contains exactly one instance of the digits from 1 to 9.



Example:

Input:

grid = { {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: Each row, column and 3\*3 box of the output matrix contains unique numbers.

Method : Backtracking.

Approach:

Like all other [Backtracking problems](https://www.geeksforgeeks.org/archives/tag/backtracking), Sudoku can be solved by one by one assigning numbers to empty cells. Before assigning a number, check whether it is safe to assign. Check that the same number is not present in the current row, current column and current 3X3 subgrid. After checking for safety, assign the number, and recursively check whether this assignment leads to a solution or not. If the assignment doesn’t lead to a solution, then try the next number for the current empty cell. And if none of the number (1 to 9) leads to a solution, return false and print no solution exists.