BACKTRACKING

Suppose we have ~~various~~ series decisions among various choices. Don't have enough info. then we try 1 option at a time, if it leads to dead end we backtrack to the parent & try other options

Some sequence of decisions (possibly >1) can give soln to given problem.

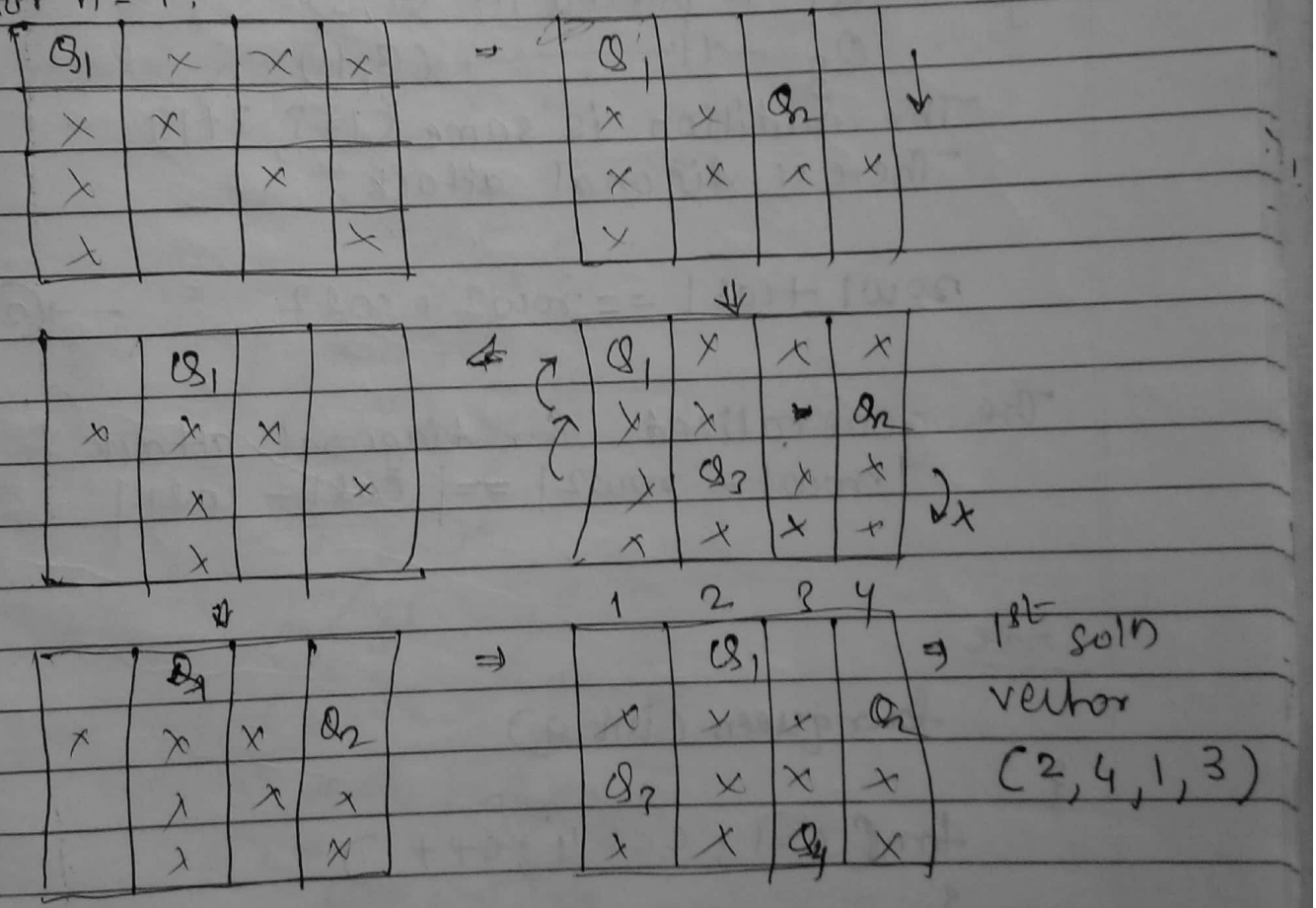
Backtracking is a methodical way of trying various sequence of decisions until we find soln that works.

Backtracking is depth first search of state space tree

N-Queen Problem

To simplify this problem we assume that queens are numbered from 1 to n & rows also. i.e Queen no. represents row no. (to avoid row attack)
∴ we need to find column position of each queen. the soln vector contains values from 1 to n. that values are column positions of each queen.

eg. for n = 4.



Implicit constraints. ← No queen has row, column & diagonal.

Explicit ← soln vector has value from 1 to n & every cell of soln vector should have unique value.

Diagonal Attack

For upper left - lower right.

eg. Q_2 is placed in (2,1) → row1 = 2, col1 = 1
Q_4 —11— (4,3) → row2 = 4, col2 = 3

The difference is same between 2 cells.
∴ There is diagonal attack.

$row1 - col1 == row2 - col2$ → (1)

For upper right → lower left.

eg. Q_1 is placed in (1,3)
Q_3 —11— (3,1)

The addition is same (1+3, 3+1)
There is digonal attack.

$row1 + col1 == row2 + col2$ → (2)

The generalized for diagonal attack

$|row1 - row2| == |col1 - col2|$

Code:

```
fourqueen (int q)
{
  for( c=1 ; c<=4 ; c++ )
  {
    if (place (q,c))
    {
      col[q] = c;
      if (q==4)
        display (col)
      else
        fourqueen (q+1);
    }
  }
}
```

```
boolean place (int q, int c)
{
  for (int k=1; k<=q-1; k++)
  {
    if (col[k]==c || (q-k)==math.abs(col[k]-c))
    {
      return false;
    }
  }
  return true;
}
```