Backtracking

- Backtracking is an approach to problem solving which is usually applied to constraint satisfaction problems like puzzles.

- In a backtracking solution, a search path is followed and algo. backtracks at a particular point (decision point) in the path as soon as it realizes that this path won't lead to a valid solution.
- Then it follows another path starting from previous decision point.
- In this way, different paths are repeatedly explored to arrive at the final solution.
- Backtracking is basically a refinement of the brute-force approach which then solution to a problem is found using systematic search than trying out all possible combination.

Backtracking

General Method

- In this desired solution is expressible as n-tuple $\langle x_1, x_2 \cdots x_n \rangle$ where x_i is chosen from some finite set S_i.
- The solution maximizes or minimizes or satisfies a criterion function on $(x_1, x_2 \cdots x_n)$.
- Problem can be categorized into 3 categories.
  i.e for a problem P, let C → set of constraints for P
  D - set containing all solutions satisfying C.

(i) Decision Problem : finding whether there is any feasible solution.

(ii) Optimization problem : What is the best solution?

(iii) Enumeration problem : Listing of all the feasible solution.

- Basic Idea : 1) To build a solution vector, by adding one component at a time.
  2) Test whether vector formed has any chance of success ie satisfies criterion (Bounding function)
  3) If partial vector generated does not lead to an optimal solution, then ignore such vector.

* Constraints

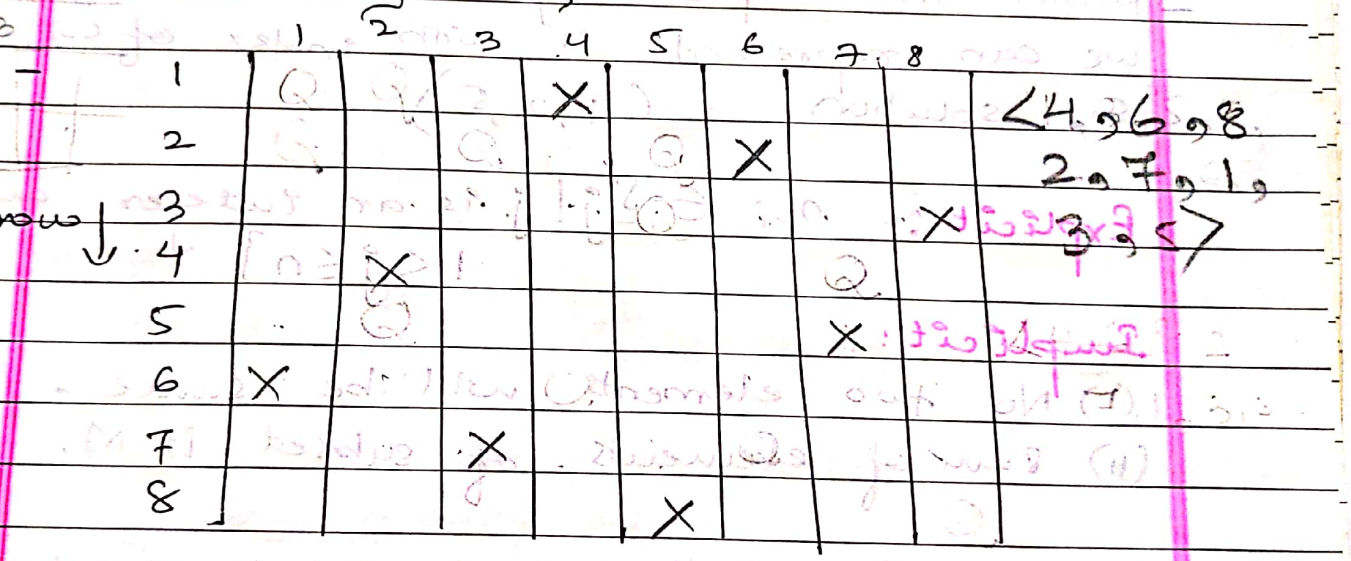
- For any problem, there are 2 types of constraints

1) Explicit constraints :
   - Rules that restricts each x_i to take on values only from a given set.
   - Explicit constraint depends on particular instance I of the problem.

- All tuples / vectors that satisfy the explicit constraints defines a possible solution space for I.

2) Implicit Constraints
- Rules that decides which tuples in the solution space I satisfy the criterion function.
- Thus, It describes the way in which x_i are related to each other.

Example 1: 8 queens problem
- Problem is to place 8 queens on 8 x 8 chessboard such that no two queens are non-attacking.



- Solutions can be represented using 8-tuples $(x_1, x_2 \ldots x_8)$, x_i is the column in which queen 'i' is placed.

- Explicit Constraint: $S_i = \{1, 2, 3, 4, 5, 6, 7, 8\}$
$1 \le i \le 8$

- Implicit Constraints:
(1) No two x_i will be same (All queens must be in different column.

(ii) No two queens can be on same row, column or diagonal.

Example 2 : Sum of subsets

- Given positive numbers w_i, $1 \le i \le n$, and m problem is to find all subsets of the w_i whose sum are m.

Eg: $n = 6$

$\Rightarrow (w_1, w_2, w_3, w_4, w_5, w_6)$

$A \Rightarrow (25, 8, 16, 32, 26, 52)$ $[M = 59]$

- possible subset = (25, 8, 26)

- Rather than representing solution vector by w_i we can represent it with index of w_i

So, solution = (1, 2, 5)

Explicit :- $x_i \in \{j \mid j$ is an integer and $1 \le j \le n\}$

Implicit :

(i) No two elements will be same.

(ii) Sum of elements of subset is M.

- Another formulation, each solution subset is represented by n-tuple $(x_1, \ldots x_n)$

Such that $x_i \in [0, 1]$, $1 \le i \le n$.

i.e $x_i = 0$ if w_i is not present else

$x_i = 1$ if w_i is present.

So, Solution = {1, 1, 0, 0, 1, 0}

Tree Organization -- Terminologies.

- Problem state : Each node in the tree.
- State space : All path from the root to other nodes.
- Solution states : those problem states s for which the path from the root to s defines a tuple in the solution space.
- Answer states : those solution states s for which the path from the root to s defines a tuple that is the member of the set of solutions.
- State space tree : This tree organization of solution space.
- Static Trees : Tree organizations are independent of the problem instances being solved.
- Dynamic Trees : Tree organisation is determined dynamically as the solution space is searched.
- Dead Node : Generate node which is not to be expanded further or all of whose children have been generated.
- Live node : A node which has been generated and all of its children have not yet been generated.

- E-node (expanded) : Live node whose children are currently being generated.

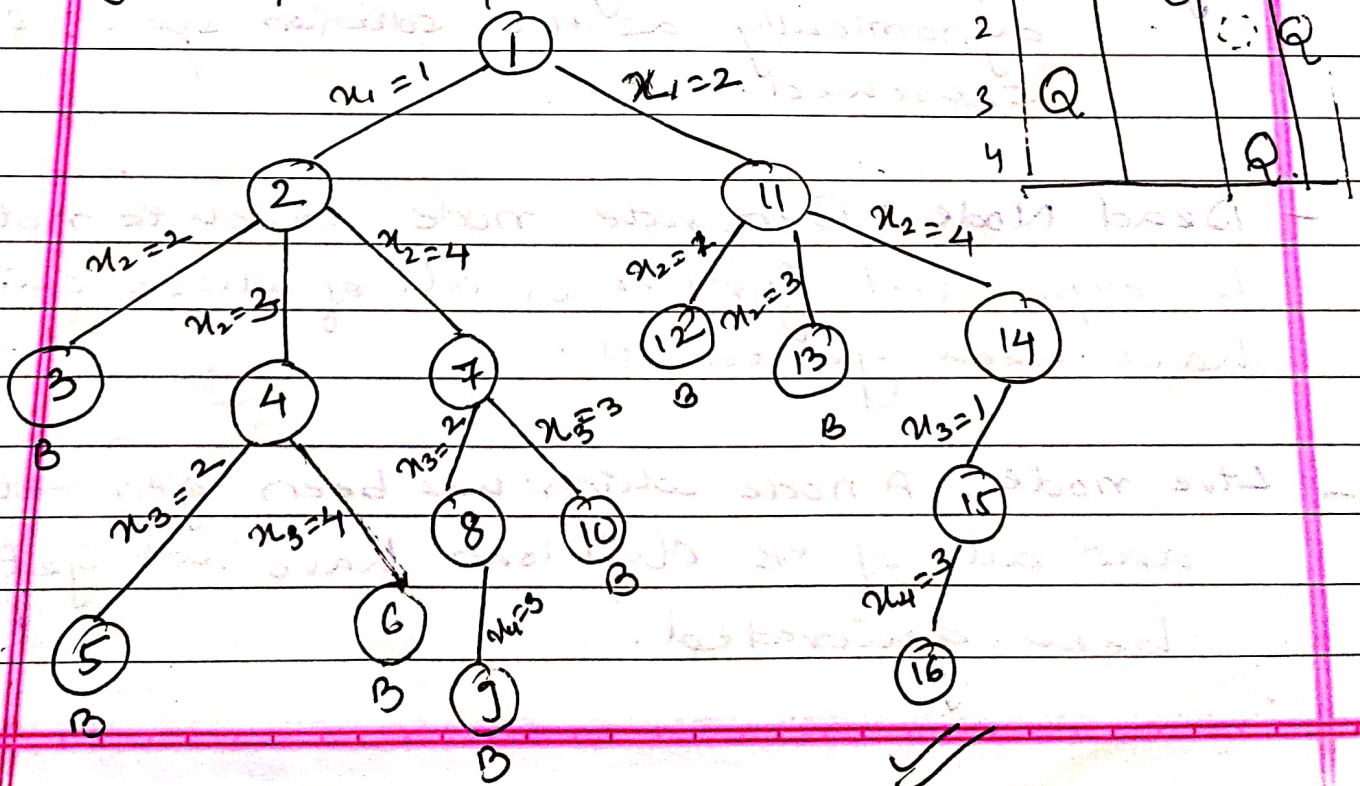
- 2 different ways for generating problem states

1] Backtracking : (Depth first node with bounding function)
- As soon as a new child 'C' of the current E-node 'R' is generated, then C becomes new E-node.
- Then R will become E-node again when the subtree C has been fully explored.

2] Branch and Bound :
- In this Enode remain E-node until it is dead.

In both method, bounding fn we use to kill live nodes.

eg 4 queen problem



$T(x_1, x_2 \ldots x_i)$ is set of all possible values for x_{i+1} such that $(x_1, x_2 \ldots x_{i+1})$ is a path to problem state.

Recursive Backtracking Algo

Algo

```
Backtrack (k)
// This schema describes backtracking process
// using recursion.
// On entering, the first k-1 values
// x[1], x[2] ..... x[k-1] of the soln vector
// x[1...n] has been assigned.
// x[] & n are global.
1. for each x[k] ∈ T(x[1], x[2] ..... x[k-1]) do
2.     if (Bk(x[1], x[2] .... x[k]) ≠ 0) then
3.         if (x[1], x[2] ... x[k] is path to an
               answer node)
4.             then write (x[1...k])
5.         if (k<n) then backtrack (k+1).
```

- This recursive version is initially invoked by Backtract (1)

* Efficiency of Backtracking algorithm
4 factors

(1) Time to generate the next x_k
(2) The no. of x_k satisfying the explicit constraints.
(3) The time for bounding functions B_i
(4) The no. of x_k satisfying the B_i

Backtracking

8-Queen Problem (N Queen Problem)

→ **Problem :-** Consider n×n chessboard & try to find all ways to place 'n' non attacking queens.

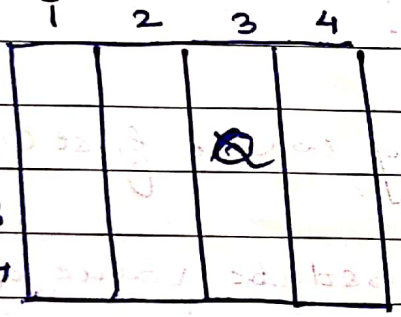
→ **Solution :-** A vector $\{x_1, x_2 \dots x_n\}$ will represents a solution in which 'x_i' is the column of the i^{th} row where the i^{th} queen is placed.

- The x_i's will be distinct since no two queens can be placed in the same column.

→ **Test whether queens are on same diagonal.**

- Let a 2-D array say $a[1 \dots n, 1 \dots n]$ represents a chess board.

For eg let n = 4



- **Upper left to lower right diagonal**

Every element on this diagonal will have same row-col value.

i.e. for $a[2,3] \Rightarrow 2-3 = -1$

Now, $a[1,2] \Rightarrow 1-2 = -1$

$a[3,4] \Rightarrow 3-4 = -1$

- **Upper right to lower left diagonal**

- Every element on this diagonal will have same row + col value

i.e for $a[2,3] \Rightarrow 2+3 = 5$

$a[4,1] \Rightarrow 4+1 = 5$

$a[3,2] \Rightarrow 3+2 = 5$

- So if two queens are placed at positions

(i,j) and (k,l), then they are on same diagonal if.

$i - j = k - l$ — ① or $i + j = k + l$ — ②

Equation ① & ② implies.

$j - l = i - k$ and. $j - l = -(i - k)$

∴ Two queens lies on the same diagonal. if and if $|j - l| = |i - k|$

→ Algorithm

```
1] Place (k, i)
   //return true if a queen can be placed in
   // kth row & ith col.
   //Return false.
   // x[] -> Global array where first (k-1) values
   //have been set.
   // Abs(r) - return absolute value of r.
   {
(k-1) 1. for j = 1 to k-1 do.
   2.   if ((x[j] = i) // Two in the same col
   3.       or (Abs (x[j] - i) = Abs (j - k)))
   4.       return false
   5. return true.
   }
```

It return true if kth queen can be placed in kth column row. It tests for both whether 'i' is distinct

from previous (x[1] .... x[k-1]) value and even for diagonal condition.

Complexity of place() method is O(k-1)

2] NQueens (k, n)
// using backtracking, this procedure prints all
// possible placements of n queens on an
// n x n chessboard so that they are
// non attacking.

{
1. for i = 1 to n do
2. if place (k, i) then
3. x[k] = i
4. if (k == n) then
5. write (x[1 ... n])
6. else NQueens (k+1, n)

The algorithm is invoked by NQueens (1, n).

→ Analysis

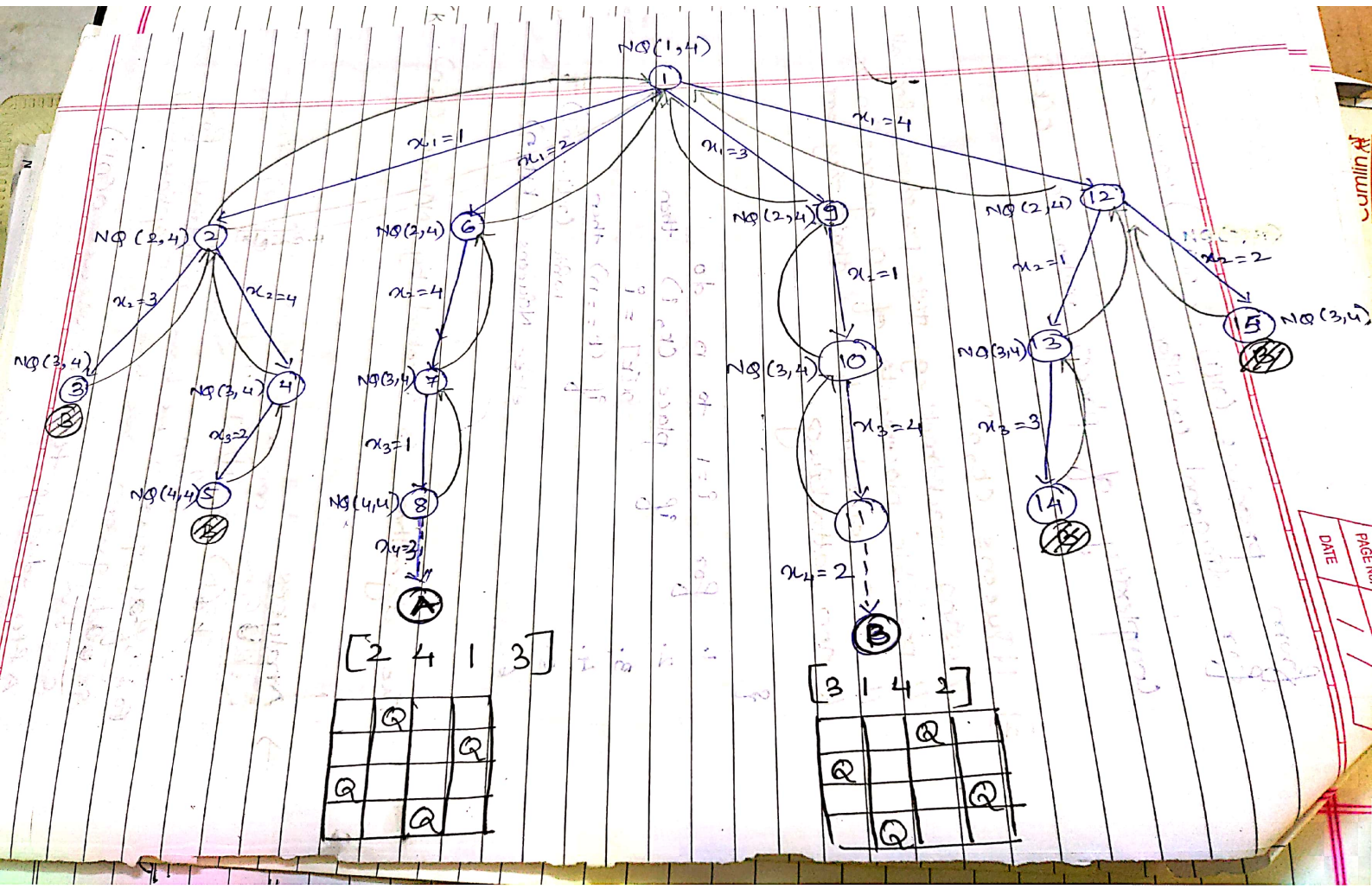
Recurrence can be defined as,

$$T(N) = N \times T(N-1) + O(N^2)$$

N → all options; T(N-1) → Rest of queens; $O(N^2)$ → constant work.

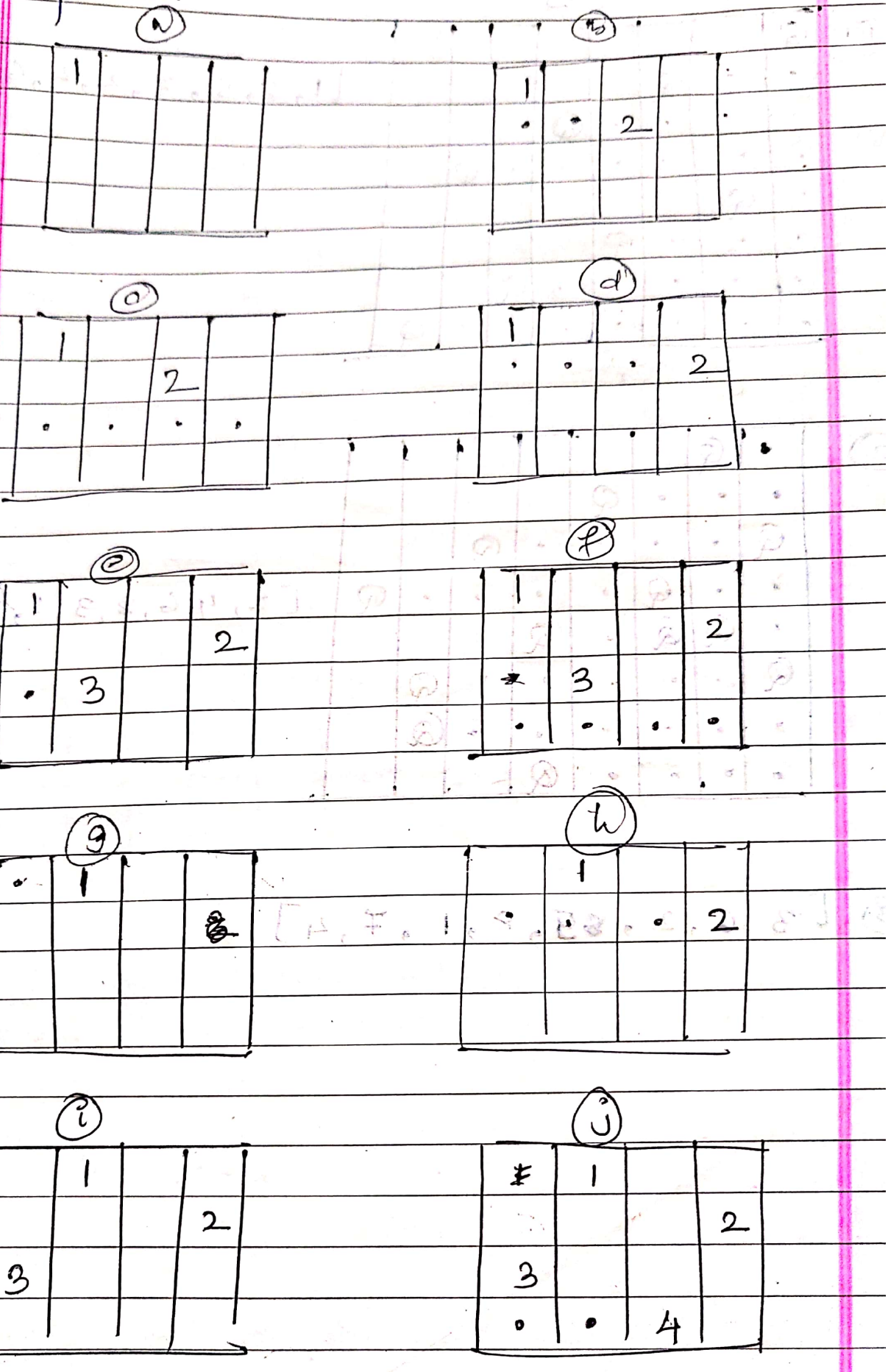
For first queen we have 'n' options, for second queen we have 'n-1' options and so on

After solving this recurrence $O(n!)$



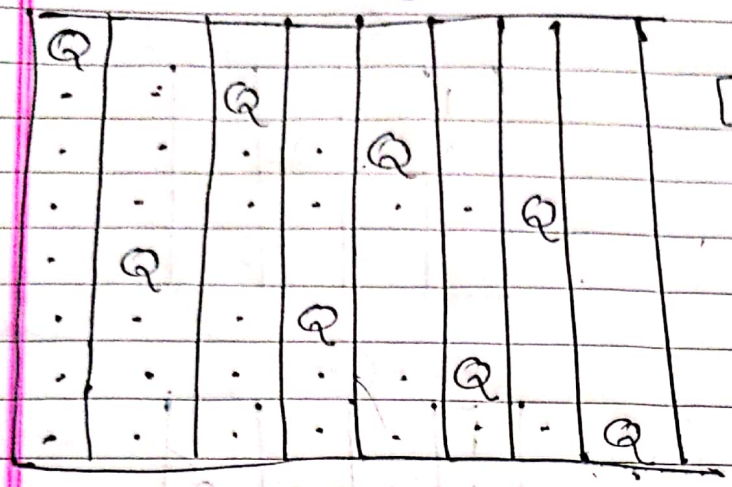
PAGE No.
DATE

Eg of backtrack solution to the 4-queens problem.



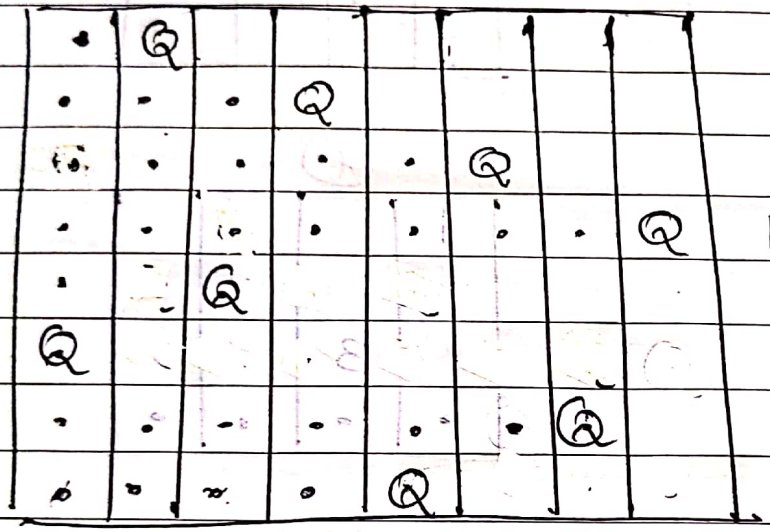
Ex: Backtracking solution for 8-queens problem

(1)



[1, 3, 5, 7, 2, 4, 6, 8]

(2)



[2, 4, 6, 8, 3, 1, 7, 5]

(3) [3, 6, 2, 5, 8, 1, 7, 4]