UNIT - 4

BACKTRACKING:

⇒ Systematic method for searching one or solutions for a given problem.

⇒ It is a refined brute force technique used for solving problems.

⇒ proposed by D.H. Lehmer and later refined by R.J. Walker

⇒ Backtracking solves multi-decision problems where the final choice leads to another set of decisions or choices

⇒ In case of impasse (deadend), one can backtrack and try other alternatives to achieve.

⇒ over all strategy may end in a successful or an unsuccessful outcome.

⇒ Backtracking can solve 3 types of problems.

① Enumeration problems

⇒ All solutions are listed for a given problem.

② Decision problems.

⇒ Solution is given in terms of yes/no.

3. Optimization problems.

⇒ Optimal solutions are required which maximize or minimize the given objective function as per the constraints of the given problem.

Eg. Real life backtracking problems.

① Searching a torch light in a dark room. If one reaches a dead end, he has to backtrack and continue the search process till the torch light is found if it is really present in the room.

BRUTE FORCE TECHNIQUE	BACKTRACKING.
⇒ Solves a given problem by listing out all its possible solutions from which the optimal solution is picked.	Solves most of the problems in polynomial time
⇒ Often leads to exponential time complexity	⇒ Solves incrementally by adding the candidate solutions till the final solution is obtained.

DOMINO PRINCIPLE.

Logic of backtracking is to construct a partial vector that consititutes a small portion of the solution of the given problem If the partial solution is not leading to a solution, then it is rejected along with its candidate solutions.

⇒ backtracking process is continued till the goal state is reached; otherwise the search is termed as unsuccessful.

BASICS OF BACKTRACKING.

⇒ It is a depth first search (DFS) with some bounding functions.

⇒ Bounding functions represent the constraints of the given problem.

⇒ 1st, the backtracking process defines a solution vector as n-tuple vector $(x_1, x_2, \ldots x_n)$ for the given problem.

$n \Rightarrow$ no. of components of the solution vector.

x_i - i ranges from 1 to n represents a partial solution.

⇒ partial solution x_i ~~is~~ are generated based on the concept of constraints.

Constraints ⇒ rules that restrict the ge[neration] or processing of a tuple.

Explicit Implicit

EXPLICIT CONSTRAINT

⇒ Rules that restrict a component of the solution vector say x_i from choosing a specific value from a set S.

Eg. $x_i \geq 0$ – explicit constraint

x_i is forced to a value ≥ 0.

IMPLICIT CONSTRAINT

Rules that limit the generation or processing of a solution vector that maximizes, minimizes or satisfies the criterion function that is also expressed as a vector $(x_1, x_2 \ldots x_n)$

⇒ Criterion function ⇒ also called as promising function or bounding or validity function.

Eg. 8 Queens problem.

⇒ 8 coins/Queens to be placed in such a manner that no two queens are in attacking position.

Constraint fn. ⇒ No 2 queens can be placed in the same row, column or diagonal

⇒ Backtracking method involves 2 stages (3)
(1) Generation of a state space tree and
(2) Exploring the state space tree.

(1) 1st stage

Generation of State Space Trees.

In 1st stage

⇒ Explicit state space tree is generated.

⇒ It is also called as solution space or recursion tree.

⇒ A state space tree is an arrangement of all possible solutions in a tree-like fashion.

↳ It can be binary tree. Every node of the state-space tree represents a partial solution that illustrates the choices made from the root to that node and the edges represent transition from states.

Terminologies

Answer States: Solution states where the path from the root to the leaf defines the solution of the problem.

Live node: A node that has been generated already but is yet to generate the children is called live node.

E-node ⇒ A node is under consideration and is in the process of being generated is called e-node.

Dead node ⇒ A node that is already explained and cannot be considered for further searches is called dead node.

(2) 2nd stage

Searching State space Trees

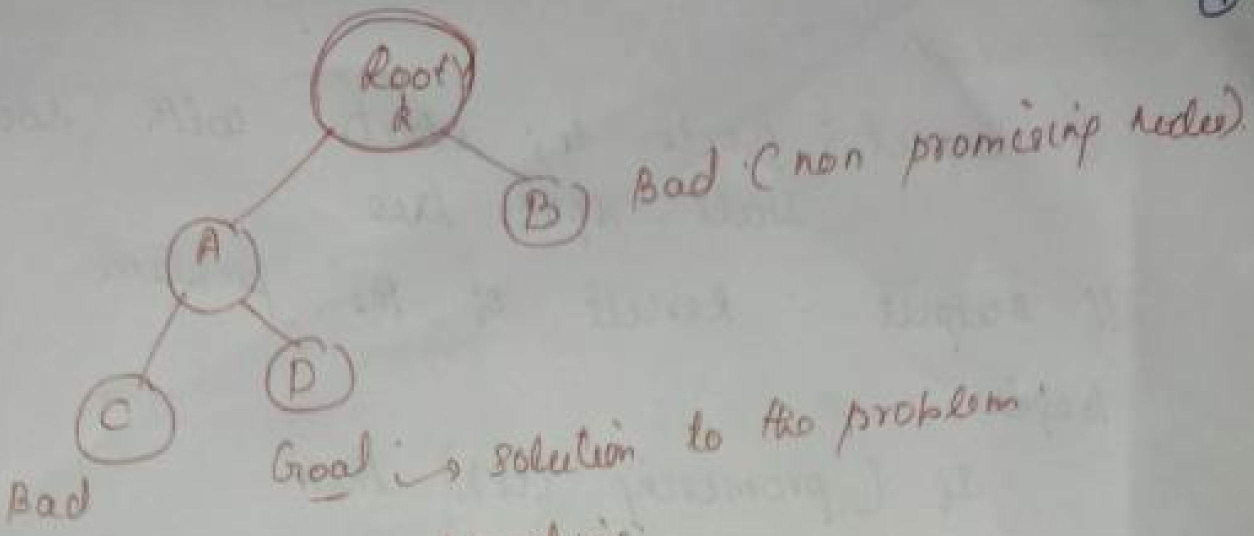
2nd stage is to explore the state space tree.

⇒ Searching strategies like BFS, DFS can be used for searching a goal in the state space tree.

⇒ Backtracking uses DFS strategy and hence it is called as refined DFS

⇒ This technique determines whether a node in the search space is 'promising' or 'non-promising'

⇒ promising node leads to final solution where non-promising node lead to a situation where solutions cannot be expected.



Eg. of backtracking

The above figure shows the backtracking approach.

1. Search starts at root R. Here there are 2 choices A & B. pick the choice A.
2. At node A, there are 2 choices C and D
3. select C, choice is bad. So backtrack to A
4. A is already explored, so move to D.
5. D is the goal state and report success
6. Backtrack to A and root R. Explore B. Since B is bad, terminate the search.

Eg. Finding the forgotten password of a suitcase.

Algorithm try (u)

// Input : node u, starts with root of the
state - space tree

// output : Result of the problem

Begin

If (promising (u)) then

if (u is a goal) then

print the solution

else

for each v, v ∈ child (u) do

try (v)

end for

endif

endif

End.

⇒ The above algorithm generates a state space tree and uses bounding functions to check whether the node is promising or not.

⇒ If the node is promising, only then it can be generated

Algorithm tryexpand (u)

// Input : Node u, starts with root of the
state - space tree

// output : Result of the problem

Begin

Generate children v of node u

for each v, v ∈ child (u) do

if (promising (u)) then

if (v is a goal) then

print the solution

else

ExpandCv)

endif

endif

endfor

End.

⇒ The above alg. starts with a root and generates children if it is promising else the alg. backtracks.

COMPLEXITY OF BACKTRACKING.

⇒ Difficult to evaluate backtracking alg. analytically.

⇒ Donald E. Knuth suggested a method where a random path can be generated from the root to a leaf of state-space tree and estimate the choices that are encountered in the path.

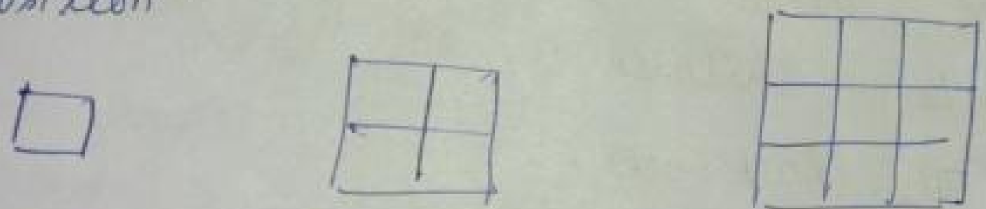
If c_1 children are encountered for the first component of the solution vector, c_2 children for the second component & so on, then the number of children encountered for the solution of random vector is given by the relation

$$T(n) = 1 + c_1 + c_1 c_2 + \dots c_1 c_2 \dots c_n.$$

N QUEEN PROBLEM

Objective

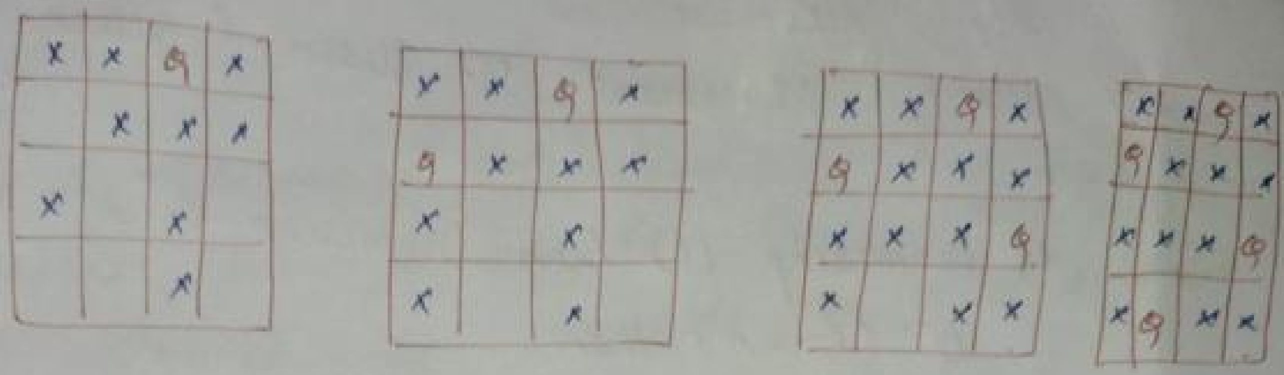
To place N queen on an $N \times N$ chessboard such that no 2 queens are in attacking position.



One queen problem   2 queen problem   3 queen problem

4 queen

STATE SPACE OF 4 QUEEN PROBLEM.



Sequence of possible solution of 4- queen problem.

⇒ A solution can be obtained only after trying all possibilities of placing the queen in columns 1 to 4.

↳ leads to 4^4 possibilities.

⇒ If the restrictions are placed that no 2 queens can be in same row / colum / diagonal, then the possibilities are reduced to $4!$

⇒ 4 queen problem can be expanded to N- queen problem. Only difference is that the state space tree for N- queen problem would be larger.

N-QUEEN PROBLEM.

Algorithm queen (i)

// I/p : queen i

// output : placement of queen i given by col(i)

Begin

if promising (i) then (promising fn is a bounding fn.)

if (i = = n) then

print col [1] .. col [n]

endif

else

for j = 1 to n do

col [i+1] = j

queen [i+1] ;

end for

endif

end

Algorithm promising (i)

// I/p : queen i

// o/p : status about the feasibility of the placement of queen i as true or false.

Begin

flag = true

for k = 1 to i-1 do

if (col [i] = col [k]) then // rows are same

if ((|col [i] - col [k]| = |i-k| or (col [i] = = col [k])) then

flag = false

endif

Endif

Endfor

End.

⇒ The promising functions check whether the queens are placed in the same row or diagonal. If so, sets the flag as false and returns it. else flag = true.

COMPLEXITY ANALYSIS

No. of nodes generated

$$1+4+4^2+\cdots+4^4 = \frac{4^{4+1}-1}{4-1}$$

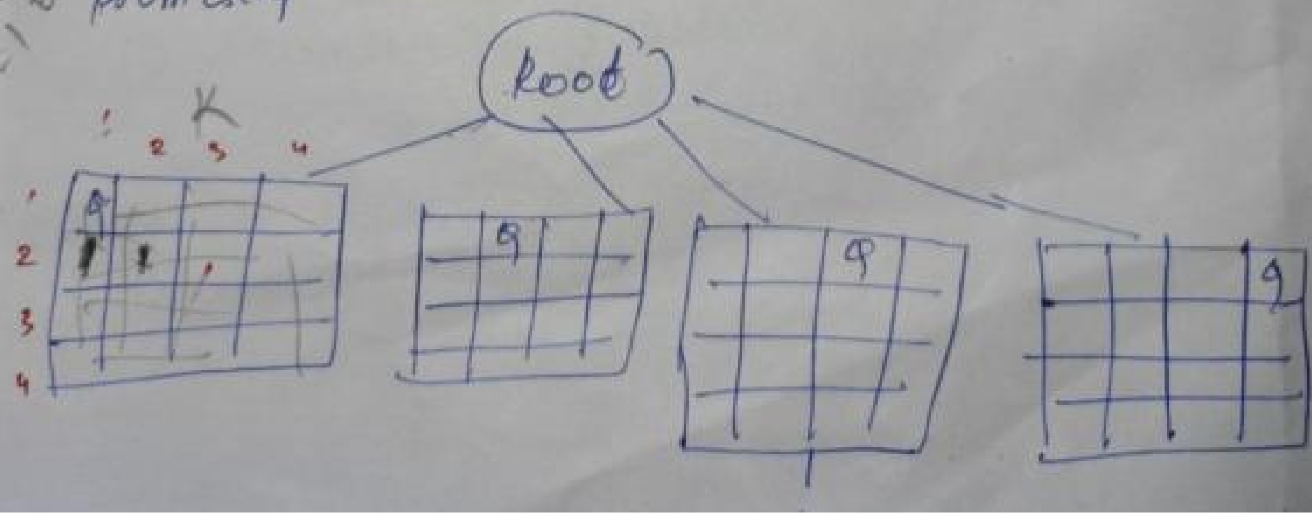
$$= \frac{4^5-1}{3}$$

Due to constraint, reduces to

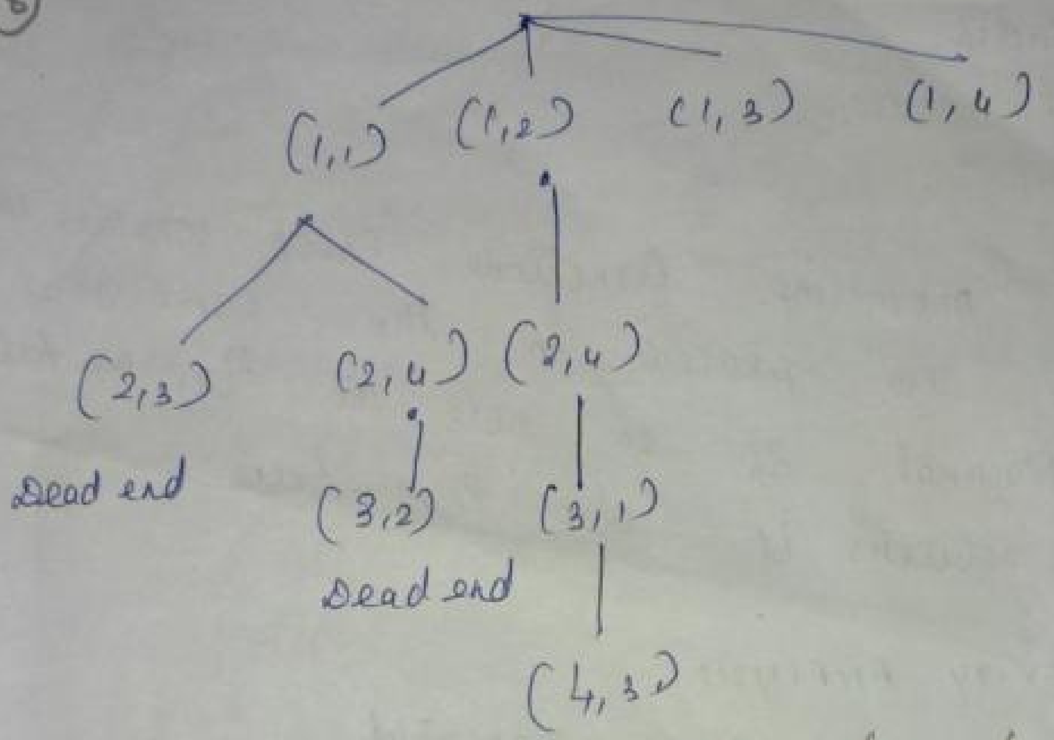
$$1+4+4\times3+4\times3\times2+4\times3\times2\times1$$

In general $1+n+n(n-1)+n(n-1)(n-2)+\cdots$ $n!$.

promising nodes are possible.



8



State space tree for 4-queens problem

3, 9, 18, 23, 25, 26, 30, 31, 34, 39, 52, 53, 59, 60, 62, 65, 66,