19/10/16

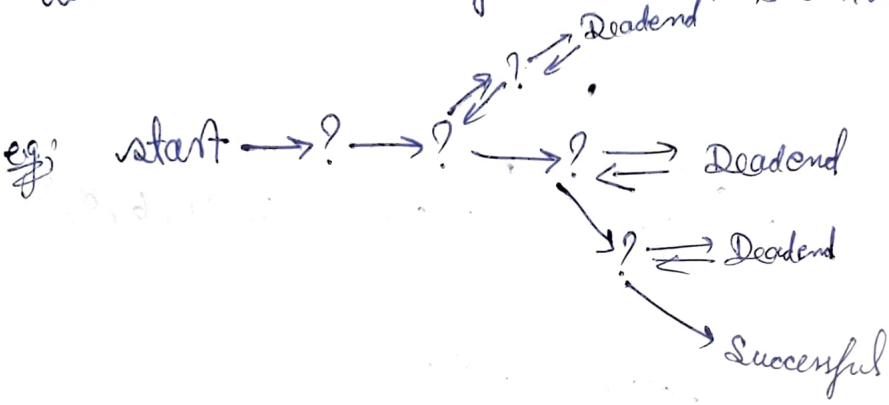
Unit - IV

Backtracking :-

Suppose we have to make a series of decision among various choices, where we do not have enough information to "know" what to choose. and each decision leads to a new set of choices.

Some sequence of choices may be a solution to our problem.

Backtracking is a method of way of trying out various sequences of decisions until we find the exact solution.



Backtracking algorithm determine problem solution by systematic searching in the solution space for given problem. This searches using a tree organisation. Each node in this tree define a problem state.

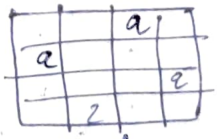
All the path from root to other node defines the state space of the problem.

Answer state are those solution states for which the path from root to leaf satisfies constraints of that problem.

eg:-

(15M)

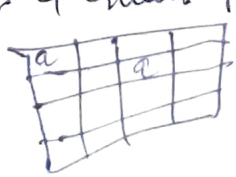
N-Queen Problem :-



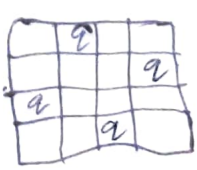
In this problem, we are given n-queens & n×n chess board.

The objective of this problem is to place all the n-queens on the chess board in such a way that no 2 queens lie on the same row, same column & diagonally.

eg:- Consider 4 Queen problem :-



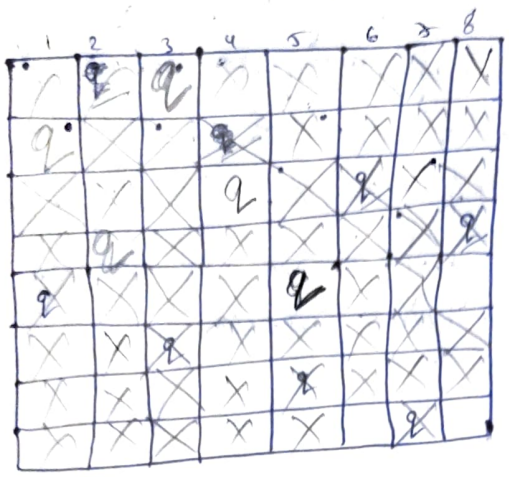
3rd queen can't be place. hence backtrack.

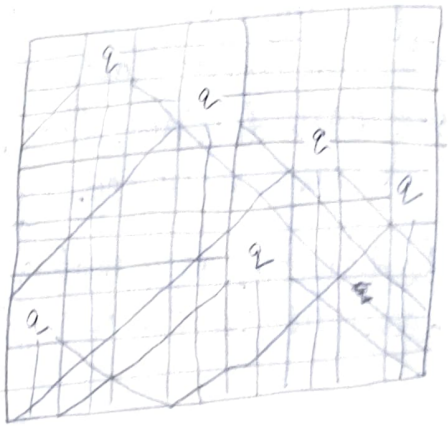


The solution is given in tuple form $\langle x_1, x_2, x_3, x_4 \rangle$

Hence the solution is: $\langle 2,4,1,3 \rangle$ (or) $\langle 3,1,4,2 \rangle$

It denote the column no of various queen.





$\langle 2,4,6,8,1,3,5,7 \rangle$ (or) $\langle 7,5,3,1,8,6,4,2 \rangle$ $[2,4,6,8,3,1,7,5]$

$\langle 8,6,4,2,7,5,3,1 \rangle$ (or) $\langle 1,3,5,7,2,4,6,8 \rangle$

Algorithm:-

Suppose that two queens are placed at position (i,j) & (k,l) therefore two queens lie on the same diagonal iff $|j-l| = |i-k|$

```
N-Queen(k,n)
1.  for i ← 1 to n
2.  do if (place[k,i] = True)
3.    {then x[k] ← i
4.       if (k=n)
5.       then for j ← 1 to n
6.            do print x[j]
7.       else N-Queen(k+1, n)
     }
```

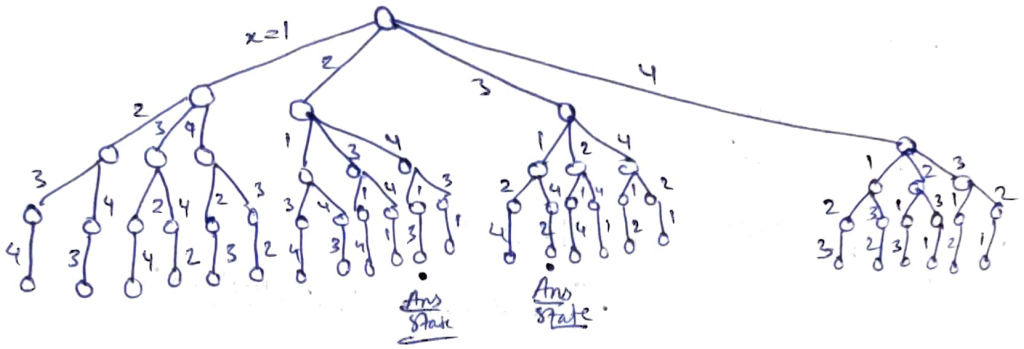
```
Bool place(k,i)                    (for same column)
1. for j ← 1 to k-1                (for diagonal)
2. do if (x[j]=i) or (abs(j-k)=abs(x[j]-i))
3.    then return false
4. return true
```

Time complexity = $\Theta(n!)$

place(k,i) : return true if k can be placed in k^{th} row & i^{th} column. otherwise it returns false.

The program is invoked by N Queen (1, n).

State space tree / solution space for 4-queen problem :-



<2,4,1,3> or <3,1,4,2>

(ISM)

Sum of Subset problem :

In this problem, we are given a set S and a number W.
We have to find the subsets of S such that the sum of elements in these subsets is equal to W.

e.g. $S = \{11, 13, 24, 7\}$ $W = 31$

fixed variable

- ✓ <3, 4> (0,0,1,1) (1, 2, 4)
- ✓ <1, 2, 4> (1,1,0,1) (3, 4)

Solution can be implemented in fixed size & variable size tuple form.

Fixed size tuple form :-

In this we use 1 or 0.
Hence in the given example, solution is : <0, 0, 1, 1) & (1, 1, 0, 1>.

Variable size tuple form :-

In this, the tuple form, the sol^n is (1, 2, 4) & (3, 4).

Algorithm :-

Assume that, all the weights are in the Ascending order.

s : the total sum of elements.
k : that element which is to be added.
r : total sum

```
Sum_of_subset(s,k,r)
1.  x[k] ← 1
2.  if( s+w[k] = w)
3   then for i ← 1 to k
4.       do print x[i]
5.  else if (s+ w[k]+w[k+1] ≤ w)
6        then Sum_of_subset(s+w[k], k+1, r-w[k])
7.  if ((s+r-w[k] ≥ w) and (s+w[k+1] ≤ w)
8.  {then x[k] ← 0
9.       Sum_of_subset(s, k+1, r-w[k])
   }
```

$$T(n) = O(2^n)$$