BACKTRACKING

TEAM MEMBERS:

AKASH SHARMA

HARSHIT GARG

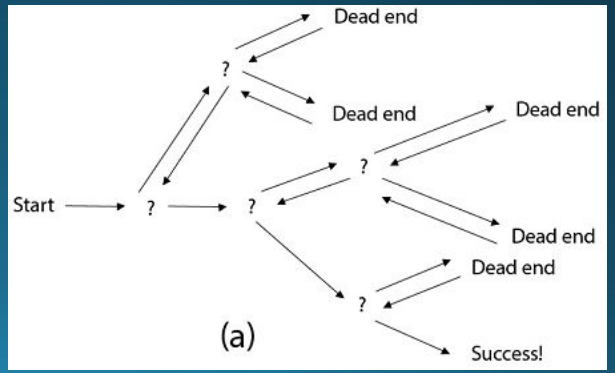
WHAT IS BACKTRACKING?

THIS IS AN ALGORITHMIC APPROACH TO FIND ONE OR MORE SOLUTION TO A PARTICULAR CATEGORY OF PROBLEMS.

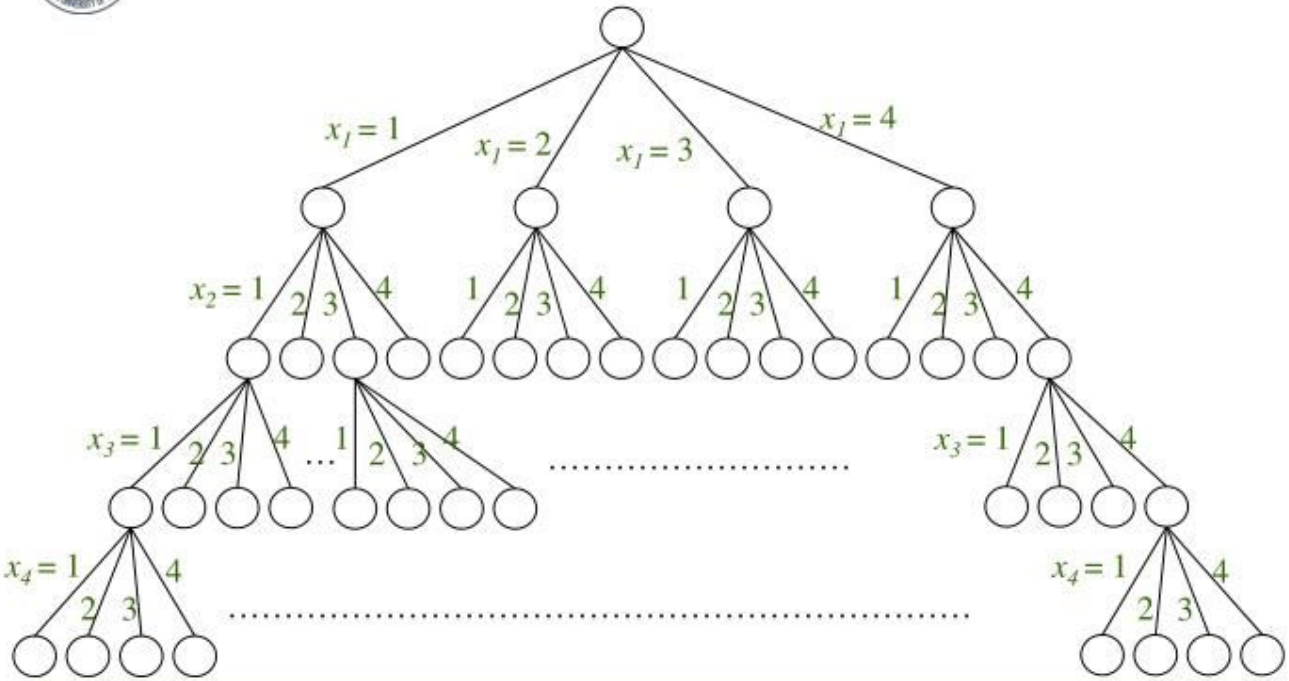
BACKTRACKING INVOLVES CHOOSING A RANDOM PATH AND EXPLORING IT UNTIL WE REACH THE GOAL, OR WE REACH A POINT OF DEAD END.

THE KEY TO BACKTRACKING INVOLVES FOLLOWING STEPS:

- CHOICE
- CONSTRAINT/ BOUNDING FUNCTION
- GOAL



State Space Tree



- State Space
  - All the paths from the root to each node

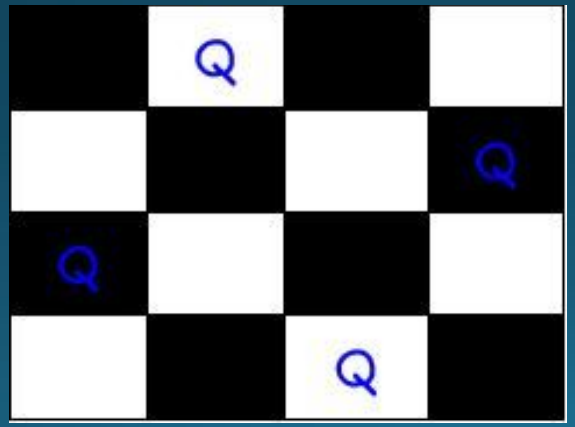
- Backtracking can be understood of as searching a tree for a particular "**goal**" leaf node.
- For any backtracking algorithm, the pseudocode is as follows:

```
boolean solve(Node n) {
        if n is a goal node, return true
        For each option O possible from n {
            if solve(O) succeeds, return true
        }
        return false
}
```

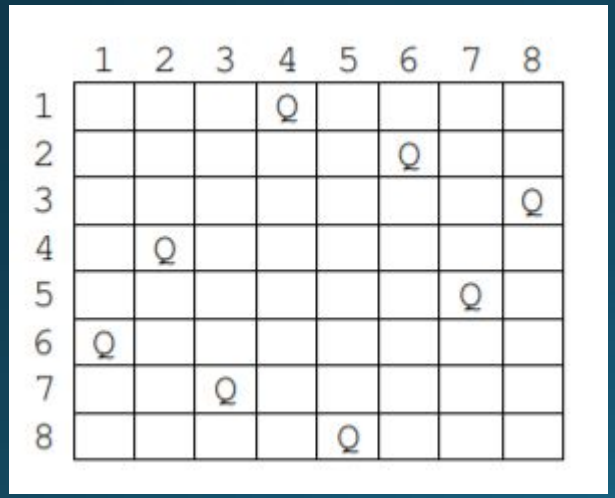
- Some applications of backtracking is in rat maze,sudoku,n queen,n knights problem,knapsack problem.

N- Queen Problem

- In N-Queen problem, we are given an NxN chessboard and we have to place n queens on the board in such a way that no two queens attack each other. A queen will attack another queen if it is placed in horizontal, vertical or diagonal points in its way. Here, we will do 4-Queen problem.



For 8-queens problem



This is one of the possible solutions for N=8 i.e. when there are 8 queens .

All the queens are placed such that they do not attack each other either diagonally, row-wise, or column-wise.

N-Queens Pseudo-code:

```
N-Queens( board[][], N )
    if N is 0                                          //All queens have been placed
        return true
    for i = 1 to N {
        for j = 1 to N {
            if is_attacked(i, j, board, N) is true
                skip it and move to next cell
            board[i][j] = 1                            //Place current queen at cell (i,j)
            if N-Queens( board, N-1) is true           // Solve subproblem
                return true                            // if solution is found return true
            board[i][j] = 0                    /* if solution is not found undo whatever changes
                                               were made i.e., remove  current queen from (i,j)*/
        }
    }
     return false
```

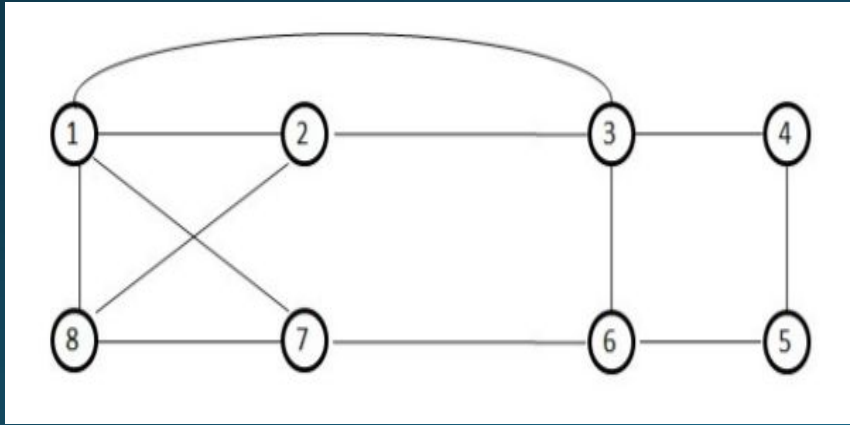
```
is_attacked( x, y, board[][], N)

    if any cell in xth row is 1                          //checking for row and column
        return true
    if any cell in yth column is 1
        return true


    if any cell (p, q) having p+q = x+y is 1             //checking for diagonals
        return true
    if any cell (p, q) having p-q = x-y is 1
        return true
     return false
```

Hamiltonian Cycle

- Hamiltonian Path in an undirected graph is a path that visits each vertex exactly once. A Hamiltonian cycle (or Hamiltonian circuit) is a Hamiltonian Path such that there is an edge (in the graph) from the last vertex to the first vertex of the Hamiltonian Path.

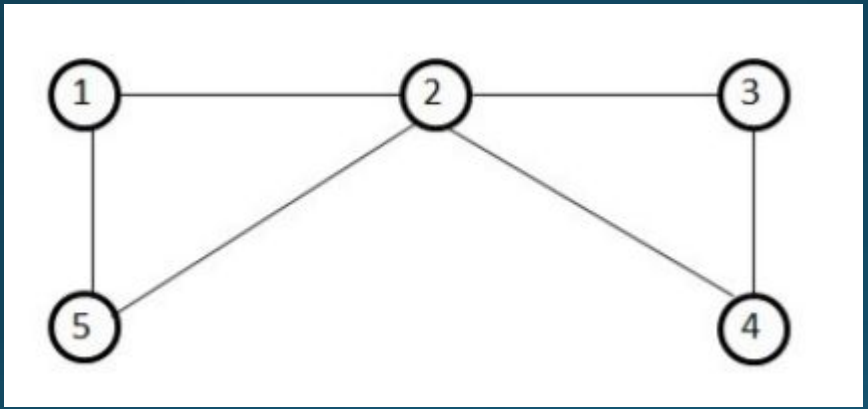


Graph containing a hamiltonian cycle

This graph contains the following cycles:

```
1,2,3,4,5,6,7,8,1
1,3,4,5,6,7,8,2,1
1,2,8,7,6,5,4,3,1
```

Hamiltonian Cycle..



This graph does not contain any hamiltonian cycle because it has a articulation point at vertex ‘2’.

Hamiltonian Cycle (contd..)

The algorithm nextvalue(k) which determines a possible next vertex for the proposed cycle.

```
Algorithm nextvalue(k)
{
 do
 {
 X[k]:=(x[k]+1)mod(n+1);
 if (x[k]=0) then    //next value is zero or not
 return;
 if (G[x[k-1],x[k]]] !=0) then    //check for edge
 {
 For j:=1 to k-1 do              //check if its duplicate or not
 if (x[j]=x[k]) then
 break;
 if (j=k) then
 if((k<n) or ((k=n)and G[x[n],x[1]]!=0))then     //check edge from last to first
 return;
 }
 }  while (true);
}
```

The algorithm Hamiltonian() uses the recursive formulation of backtracking to find all the Hamiltonian cycles of a graph.

```
Algorithm Hamiltonian(k)
 {
do
 {
 nextvalue(k);
if (x[k]=0) then return;
if (k=n) then write (x[1:n]);          //check if its last node
 else Hamiltonian(k+1);    //if not do recursively for nxt
}
while(true);
 }
```

THANK YOU