- N-Queens
- Solve SudoKu
- Word Break

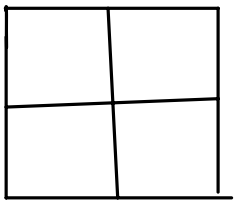
} Today's Agenda

N-Queens

Given N. chessboard of dimensions N*N.

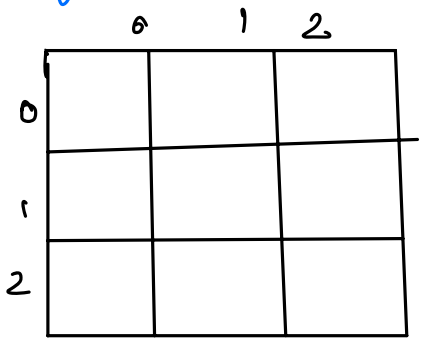
N queens → such that no queen is killing another queen. [print all valid configurations]

N=2.



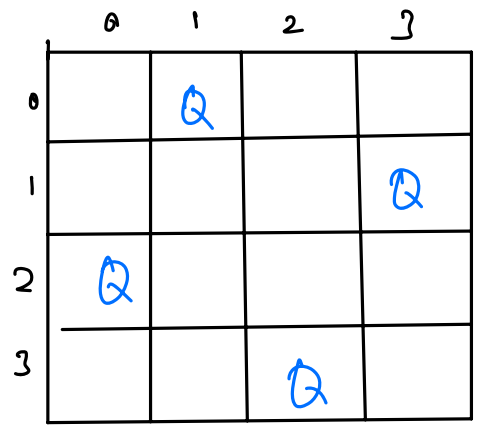
impossible.

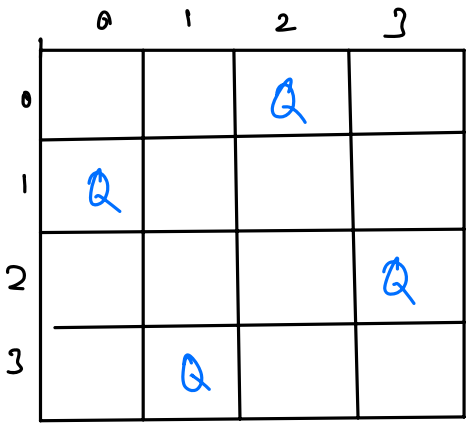
N=3



impossible.

N=4



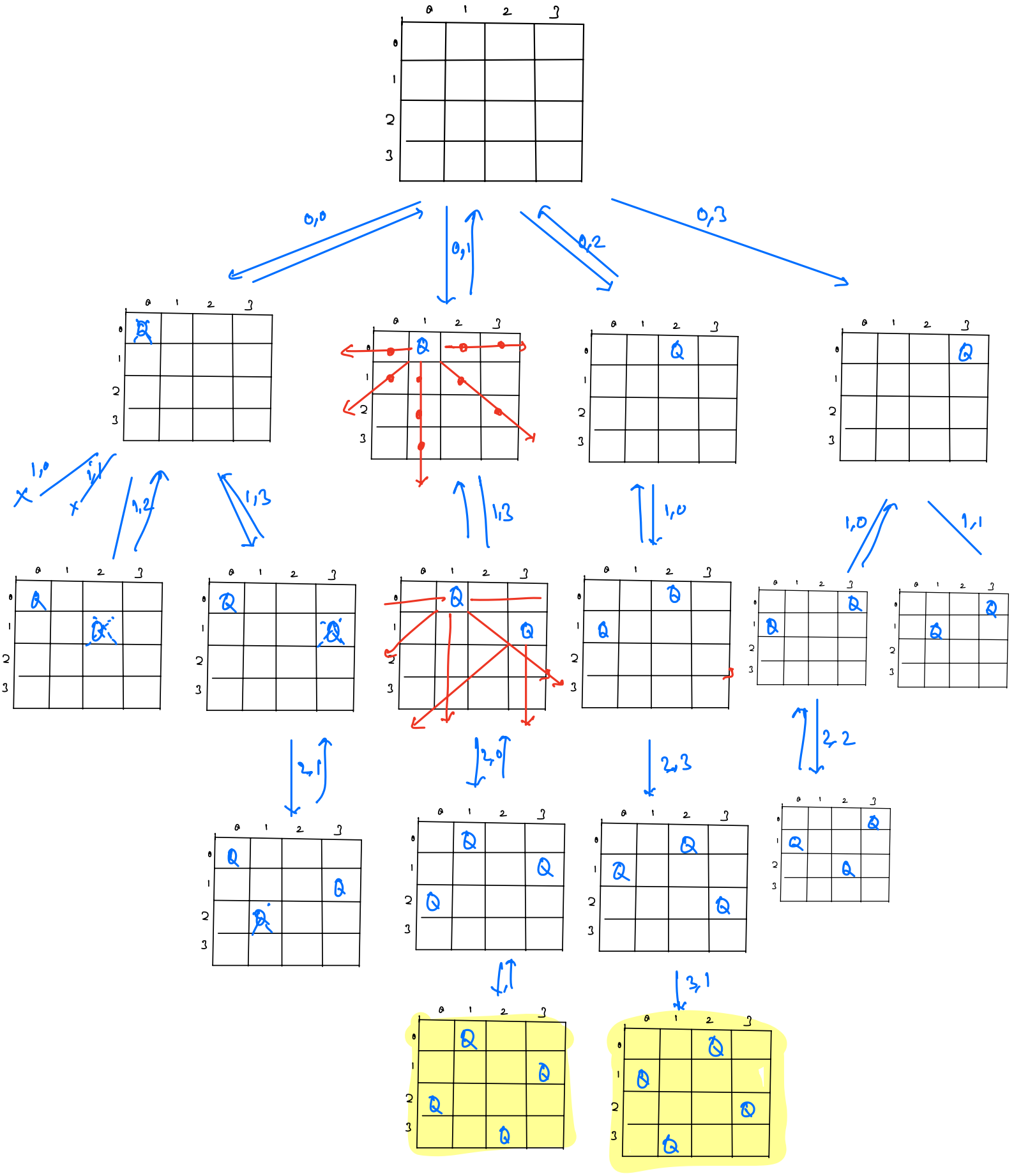


am → 0-1, 1-3, 2-0, 3-2

0-2, 1-0, 2-3, 3-1

observation → There can be only 1 queen in every row/column.

Place N Queens - try placing them row by row or column by column



code →

```
                                  0
                                  ↓
void nQueens( mat[N][N], row, N){
    if(row == N) { print(mat[7][7], return }

    for(int col=0; col<N; col++){
        if( isQueenSafe(mat[7][7, row, col) == true){
            mat[row][col] = 'Q';
            nQueens(mat[7][7, row+1, N);
            mat[row][col] = '.';
        }
    }
}
```

```
boolean isQueenSafe( mat[N][N], int row, int col){      T.C → O(N)
    for( i=0; i<row; i++){
    {   if(mat[i][col] == 'Q'){ return false }
    }

    for( i=row-1, j=col+1; i≥0 && j<N; i--, j++){
    [   if(mat[i][j] == 'Q'){ return false }
    }

    for( i=row-1, j=col-1; i≥0 && j≥0; i--, j--){
    [   if(mat[i][j] == 'Q'){ return false }
    }

    return true;
}
```

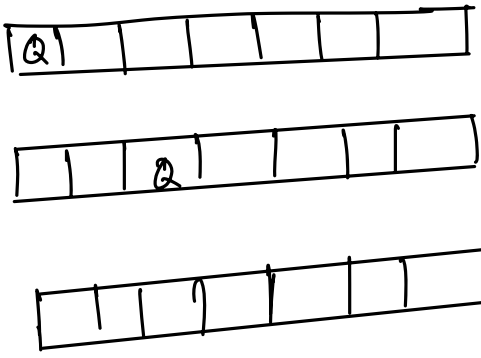
0th row → N

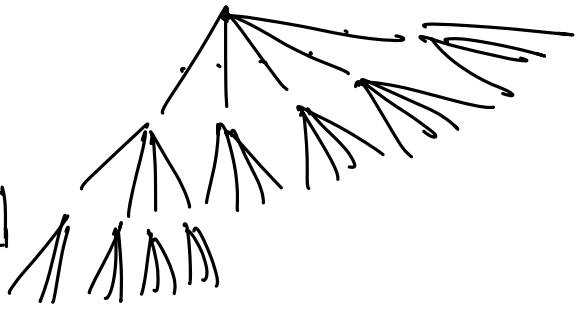
1st row → N-1

2nd row → N-2

3rd row → N-3

4th row → N-4





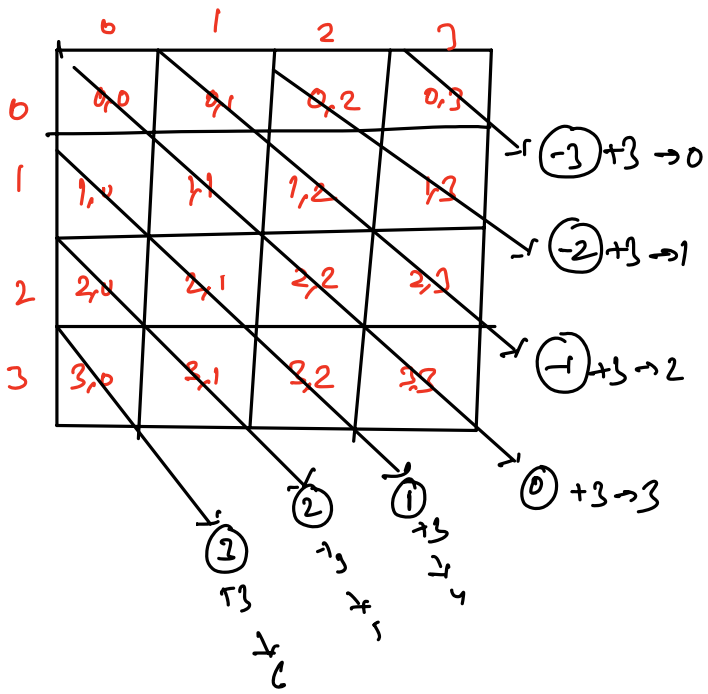
T.C → O(N × N!)

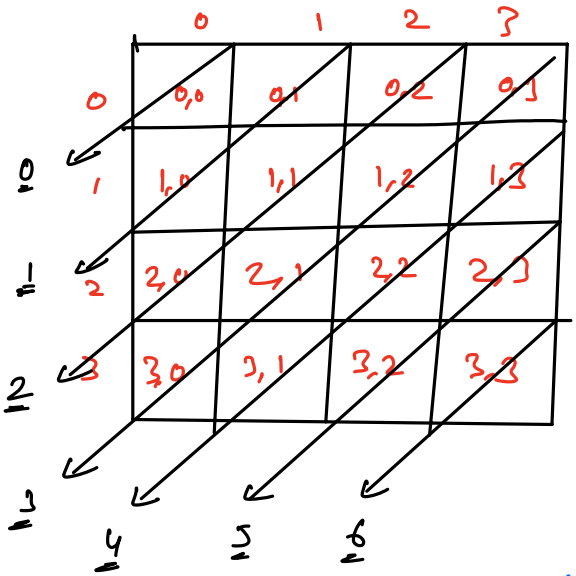
S.C → O(N² + N)

matrix

recursive stack.

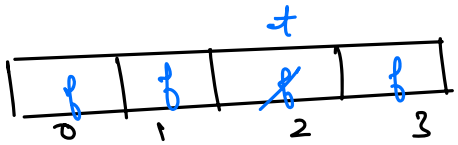
i - j + (N-1)





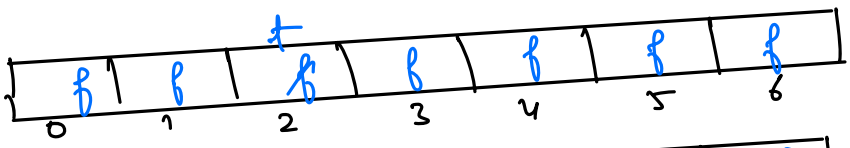
row 0! 0,2.

cols →

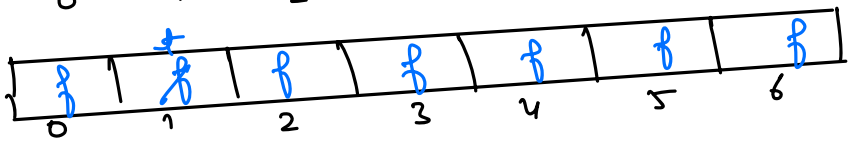


check → [2,1]

anti_diag[2N-1] →



diag [2N-1] →



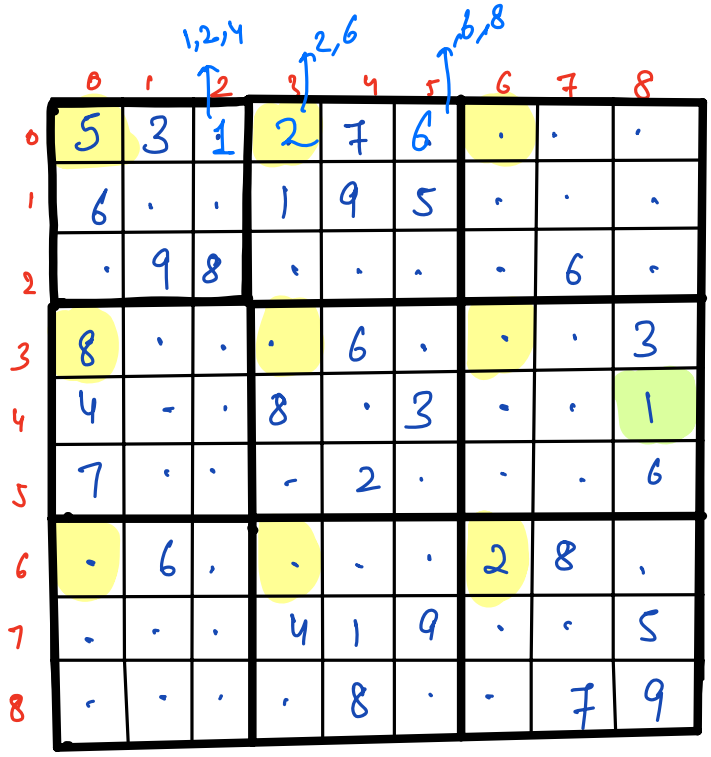
#code→

```
void nQueens( mat[N][N], i, N, cols[N], diag[2N-1], anti-diag[2N-1]){
    if( i == N) { print(mat[7[7), return }
    for( j=0; j<N; j++){
        if( cols[j] == false && antidiag[i+j] == false &&
            diag[i-j+N-1] == false ){
            mat[i][j] = 'Q';
            cols[j] = true;
            diag[i-j+N-1] = true;
            antidiag[i+j] = true;
            nQueens( mat[7[7, i+1, N, cols, diag, anti-diag);
            mat[i][j] = '.';
            cols[j] = false;
            diag[i-j+N-1] = false;
            antidiag[i+j] = false;
        }
    }
}
```

T.C → O(N!)
S.C → O(N²)

Sudoku

Given a partially solved state of sudoku. Find the solution of sudoku. [1 unique solution exists]



N=9.

Rules for sudoku -

- Each row must contain all the numbers from 1 to N.
- Each column must contain all the numbers from 1 to N
- Each $\sqrt{N} \times \sqrt{N}$ grid must contain all the numbers from 1 to N.

4, 8.

↳ closest multiple of 3 which is ≤ 4 → 3

↳ closest multiple of 3 which is ≤ 8 → 6

i → closest multiple of $\sqrt{N}$ which is $\leq i$ ⇒ $i - (i \% \sqrt{N})$

j → closest multiple of $\sqrt{N}$ which is $\leq j$ ⇒ $j - (j \% \sqrt{N})$

code :-

```
                                    0   0
                                    ↑   ↑
boolean sudoku (mat[N][N], i, j, N) {
    if( j == N){
        j = 0, i = i+1;
    }
    if( i == N){
        return true;
    }
    if (mat[i][j] != '.'){
        if (sudoku( mat[][], i, j+1, N) == true){
            return true;
        }
    }
    else{
        for( x=1; x <= N; x++){
            if( isValid ( mat[N][N], i, j, x) == true){
                mat[i][j] = x;
                if ( sudoku( mat[][], i, j+1, N) == true){
                    return true;
                }
                mat[i][j] = '.';
            }
        }
    }
    return false;
}
```

TC → $O(N^{\text{no. of empty spots}})$

SC → $O(N^2)$

```
boolean isValid( mat[N][N], row, col, x ) {        → O(N)

    for( j=0; j<N; j++){
    {   if( mat[row][j] == x) { return false }
    }

    for( i=0; i<N; i++) {
    {   if( mat[i][col] == x) { return false }
    }

    row → row - (row % √N);
    col → col - (col % √N);          } indices of top-left corner of the grid.

    for( i=0; i<√N; i++){
    {   for( j=0; j<√N; j++){
        {   if( mat[row+i][col+j] == x){
            {   return false;
            }
        }
    }

}   → return true;
```

3, 6

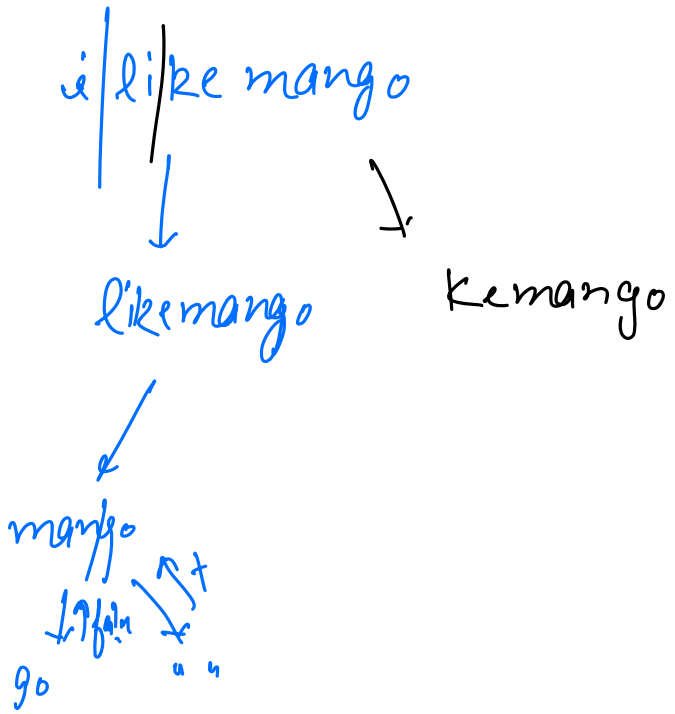
3+0, 6+0	3+0, 6+1	3+0, 6+2
3+1, 6+0	3+1, 6+1	3+1, 6+2
3+2, 6+0	3+2, 6+1	3+2, 6+2

Word Break

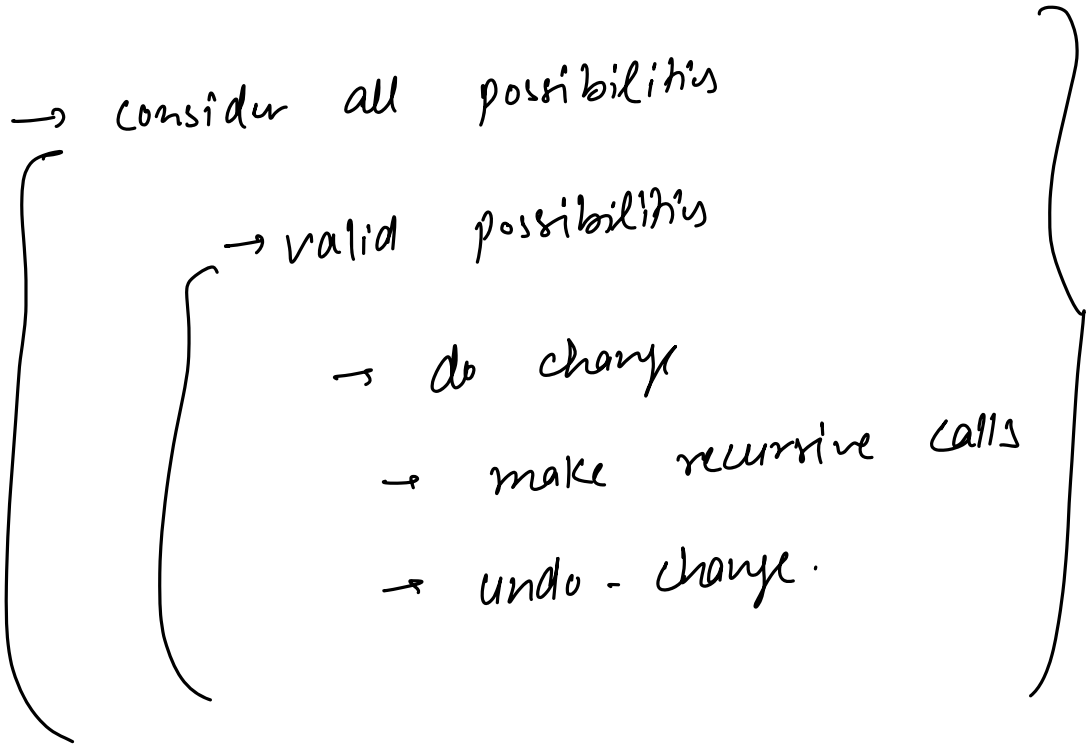
Given a dictionary of words (strings) and a string A. Check if it is possible to break A into valid words from the dictionary.

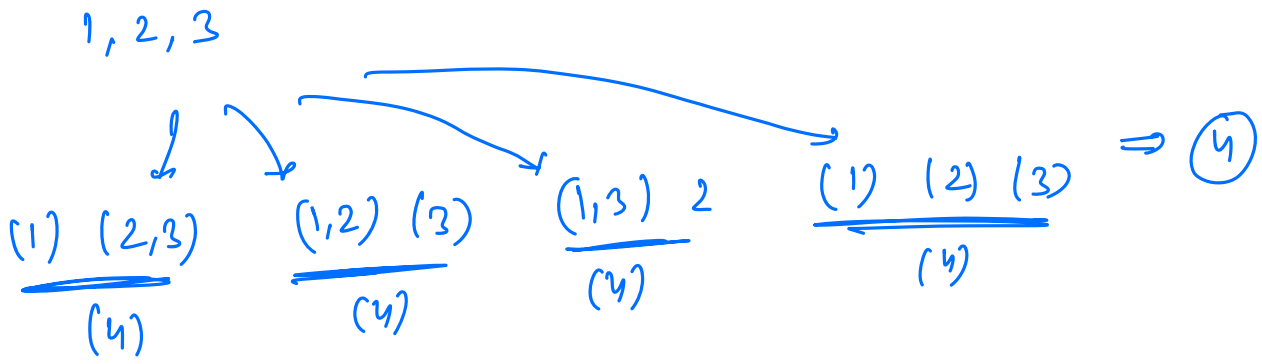
Dict → { "i", "like", "mango", "man", "gone", "ili" }

A → ilikemango



→ General structure. (all possibilities)





1, 2, 3, 4

$$way(n-1) + (n-1) \times way(n-2)$$

$$dp[i] = dp[i-1] + (i-1) \times dp[i-2]$$