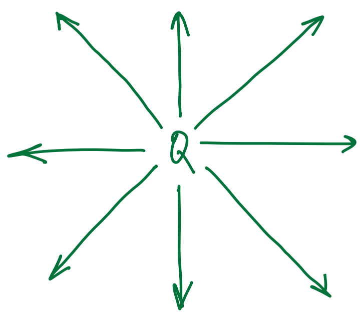
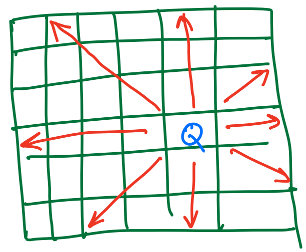
- N Queens
- Sudoku
- Word Break

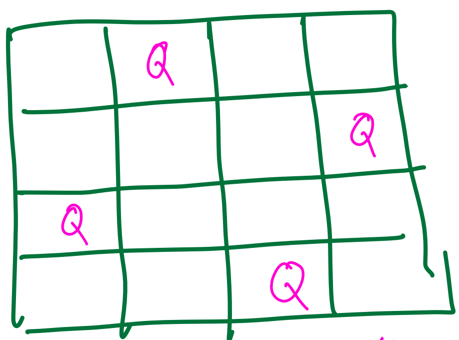
Q1) Given an N*N chessboard and N queens, arrange the queens in a way that no queen can attack the other queens.





N=1

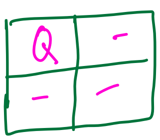
N=4



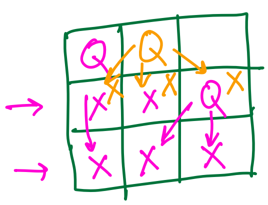
In each row → 1 queen

Row by row → Select the pos^n of queen in each row.

N=2



N=3



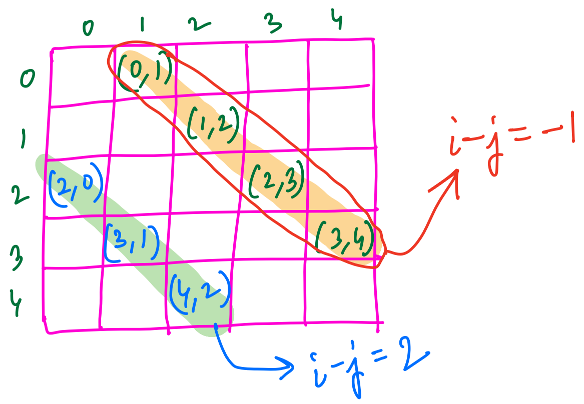
Each row ⇒ we'll choose 1 col as posⁿ.

→ Col ⇒ 0 to n-1

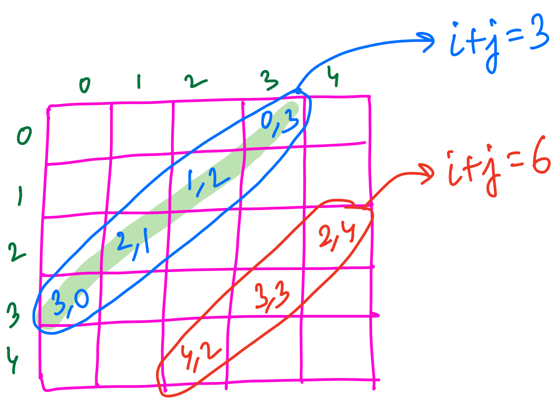
→ Ensure ⇒ No col has 2 queens ⇒ Hashset of column nos.

→ Ensure ⇒ No diagonal has 2 queens ⇒ Hashset of diag nos. $(i-j)$

→ Ensure ⇒ No anti-diagonal has 2 queens. ⇒ Hashset of anti-diag. nos. $(i+j)$

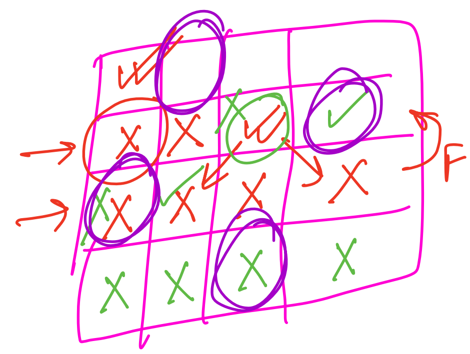


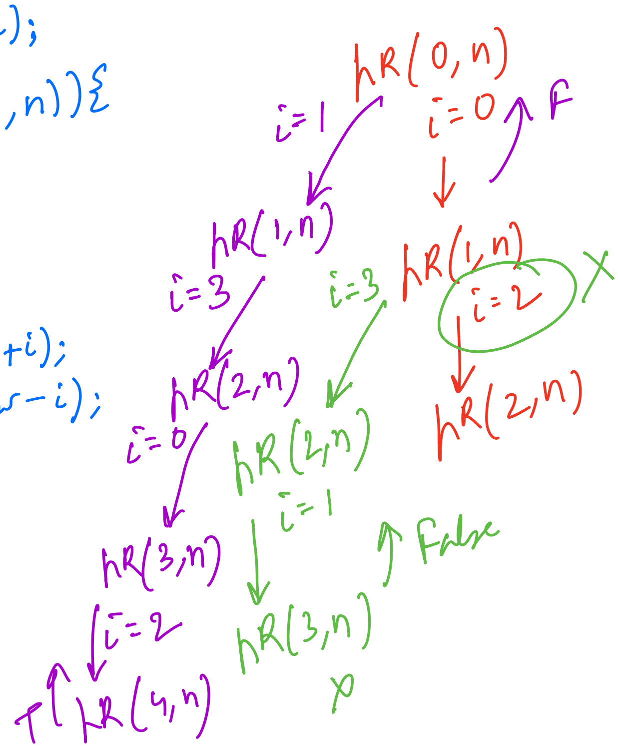
diagonal number ≡ $\boxed{i-j}$

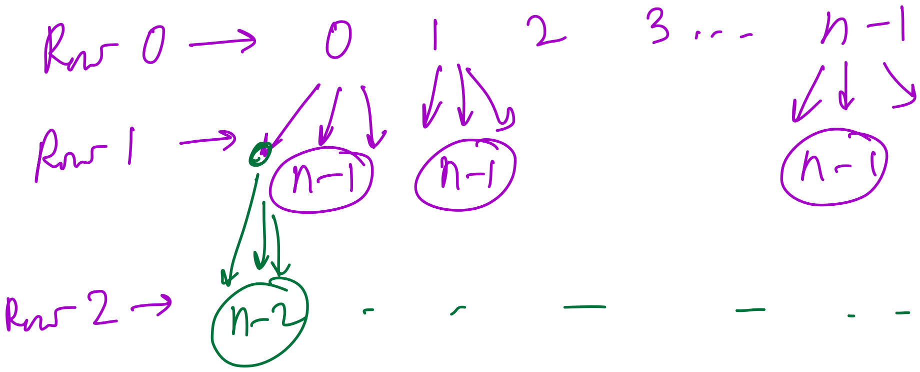


anti-diagonal number ⇒ $\boxed{i+j}$

```java
HashSet<Integer> cols = new HashSet<>();
HashSet<Integer> sumDiag = new HashSet<>();
HashSet<Integer> diffDiag = new HashSet<>();
int mat[][] = new int[n][n];
placeRow(0, n);

boolean isValid(r, c){
    return ((!cols.contains(c)) && (!sumDiag.contains(r+c)) && (!diffDiag.
                                                        contains(r-c)));
}
boolean placeRow(int row, int n){
        if(row==n){
            return true;
        for(i=0; i<n; i++){   // each col.
            if(isValid(row, i)){
                mat[row][i] = 1;
                cols.add(i);
                sumDiag.add(row+i);
                diffDiag.add(row-i);
                if(placeRow(row+1, n)){
                    return true;
                }
                mat[row][i] = 0;
                cols.remove(i);
                sumDiag.remove(row+i);
                diffDiag.remove(row-i);
            }
        }
        return false;
}
```







$$\underset{R0}{n} * \underset{R1}{(n-1)} * \underset{R2}{(n-2)} * (n-3) * \cdots * \underset{R_{n-1}}{1}$$

$$= O(n!)\ T.C$$

$O(n^2) \longrightarrow$ Total S.C

$O(n) \longrightarrow$ Extra SC (apart from the result matrix)

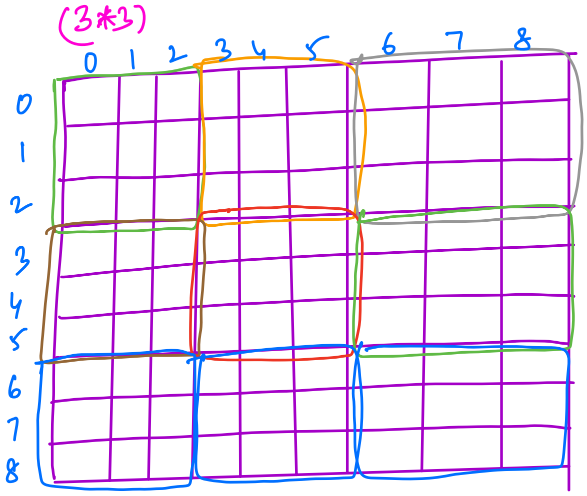
Q2) Given an 9 * 9 board, each cell contains numbers 1-9.

Some cells are empty, fill according to the following rules:-

→ Each row must contain 1-9 w/o repetitions

→ Each col must contain 1-9 w/o repetitions

→ Each block must contain 1-9 w/o repetitions
(3*3)



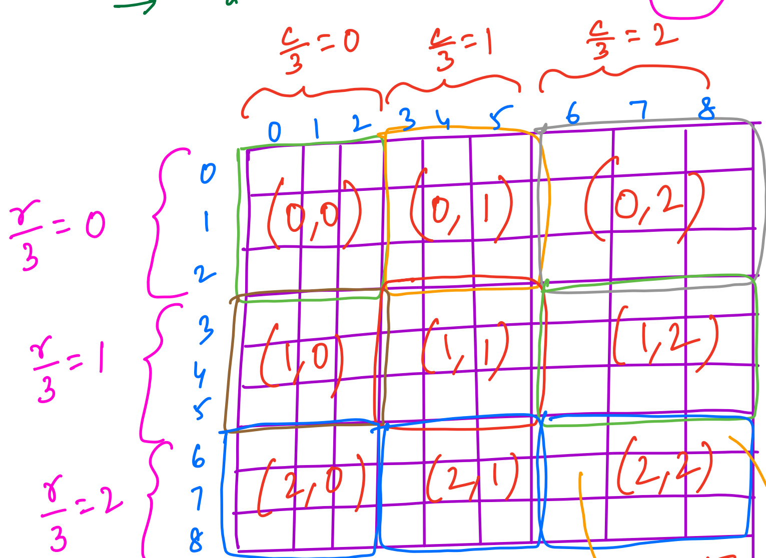
→ Move through each cell, select a number (1-9) for that cell

→ Checks :-

→ Some number x exists in current row or not → Set for each row. HashSet[9];

→ " " " " " " col " " → " " " "

→ " " " " " " block " " → " " " "



$\left(\frac{r}{3}, \frac{c}{3}\right)$

$\frac{r}{3} \in [0,2]$

$\frac{c}{3} \in [0,2]$

$\boxed{\text{blockNum} = \left(\frac{r}{3}\right) * 3 + \frac{c}{3}}$

$\left(\frac{7}{3}\right) * 3 + \frac{6}{3} = \boxed{8}$

$\left(\frac{6}{3}\right) * 3 + \left(\frac{8}{3}\right) = \boxed{8}$

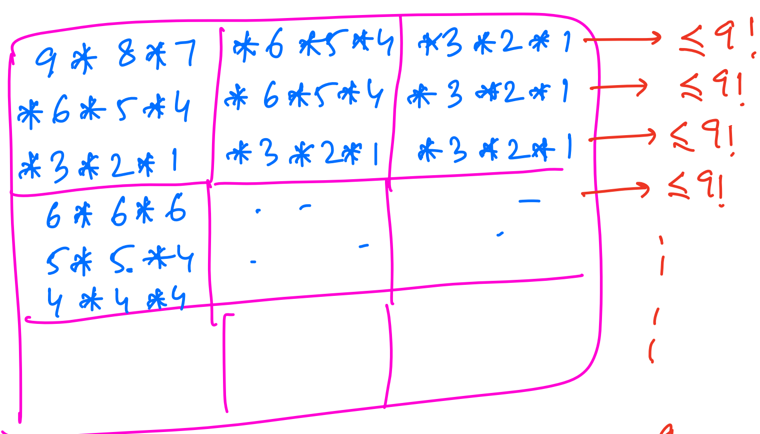
```
HashSet[ ] row = new HashSet[9];
for(i=0; i<9; i++)
    row[i] = new HashSet<>();
HashSet[ ] col = new HashSet[9];
    . . . .
HashSet[ ] blockNum = new HashSet[9];
    . . .

boolean sudoku(mat, i, j){
    if(j==9)
        return sudoku(mat, i+1, 0);
    if(i==9)
        return true;

    block_num = (i/3)*3 + (j/3);

    if(mat[i][j] != 0){
        int k = mat[i][j];
        if(row[i].contains(k) || col[j].contains(k) || blockNum[block_num].
                                                         contains(k))
            return false;

        row[i].add(k);
        col[j].add(k);
        blockNum[block_num].add(k);

        return sudoku(mat, i, j+1);
    }
```

$$\left.\begin{array}{l}(1/3)*3+(0/3)=0\\(2/3)*3+(2/3)=0\end{array}\right\}$$

```
for(k=1; k<=9; k++){
    if(row[i].contains(k) || col[j].contains(k) || blockNum[block_num].
                                                        contains(k)){
        continue;
    }
    mat[i][j]=k;
    row[i].add(k);
    col[j].add(k);
    blockNum[block_num].add(k);
    if(sudoku(mat, i, j+1))
        return true;
    mat[i][j]=0
    row[i].remove(k),
    col[j].remove(k);
    blockNum[block_num].remove(k);
}
return false;
}
```

$\underbrace{9*9*9*\ldots 9}_{81 \text{ cells}} = 9^{81}$

$9^{9*9} \rightarrow 9^{81}$



$T.C \rightarrow (9!)^9$

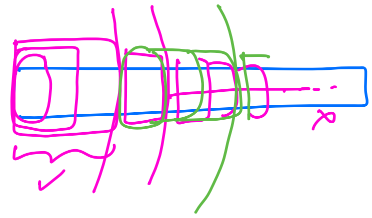
$S.C \rightarrow 9^2$.

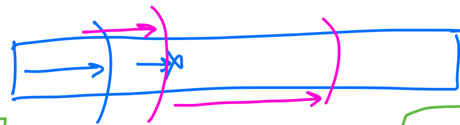
Q3) Given a dictionary of words, and a string A, check if it is possible to break A into valid words from the dictionary.

dict → {"i", "like", "man", "mango", "go", "manner"}

A = iammango ✗

A = i like mango ✓





A = "i manner go" ✓

i → man → nergo ✗

manner → go ✓

start at 0.

```
boolean PossibleToBreak(s, i) {
    if (i == s.length())
        return true;
    String str = "";
    for (j = i; j < s.length(); j++) {
        str += s[j];
        if (dict.contains(str) && PossibleToBreak(s, j+1))
            return true;
    }
    return false;
}
```

a b c d e

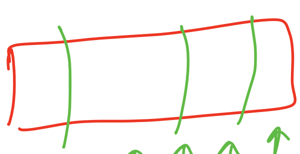
F F T

$2^n * n$

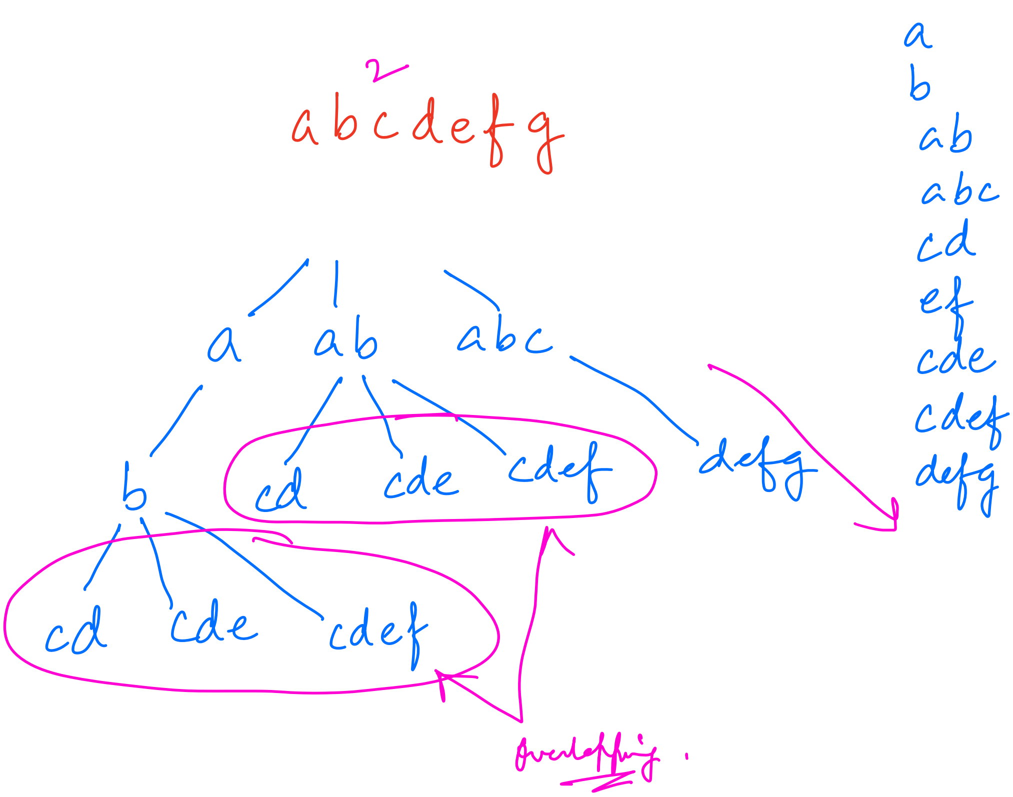
ways of breaking the word

$O(2^n * n)$ T.C.

$O(n)$ S.C



2 2 2 2 2 2 → 2^{n-1}




```
boolean dp[n] = {true};
boolean PossibleToBreak(s, i) {
    if (i == s.length())
        return true;
    if (dp[i] == false)
        return false;
    String str = "";
    for (j = i; j < s.length(); j++) {
        str += s[j];
        if ((dict.contains(str) && PossibleToBreak(s, j+1))
            return true;
    }
    dp[i] = false;
    return false;
}
```

Hashing → O(n) time

For each i → n times/iterations

* O(n) for hashing.

TC → $O(n^3)$

SC → $O(n)$

$$(a^{b!}) \% m$$

$$= \left(a^{b!-(m-1)} * a^{m-1} \% m\right) \% m$$

$$1$$

$$a^{b! - k(m-1)}$$

$$\boxed{a^{b! \% (m-1)}} \% m .$$

If m is prime, $a < m$,

$$a^{m-1} \% m = 1$$

$$a^{k(m-1)} \% m = 1$$

$$\text{ways}(n) \xrightarrow{A} \text{ways}(n-1)$$

$$\text{ways}(n) \xrightarrow{P} {}^{n-1}C_1 * \text{ways}(n-2)$$

$$\Downarrow$$

$$(n-1)$$