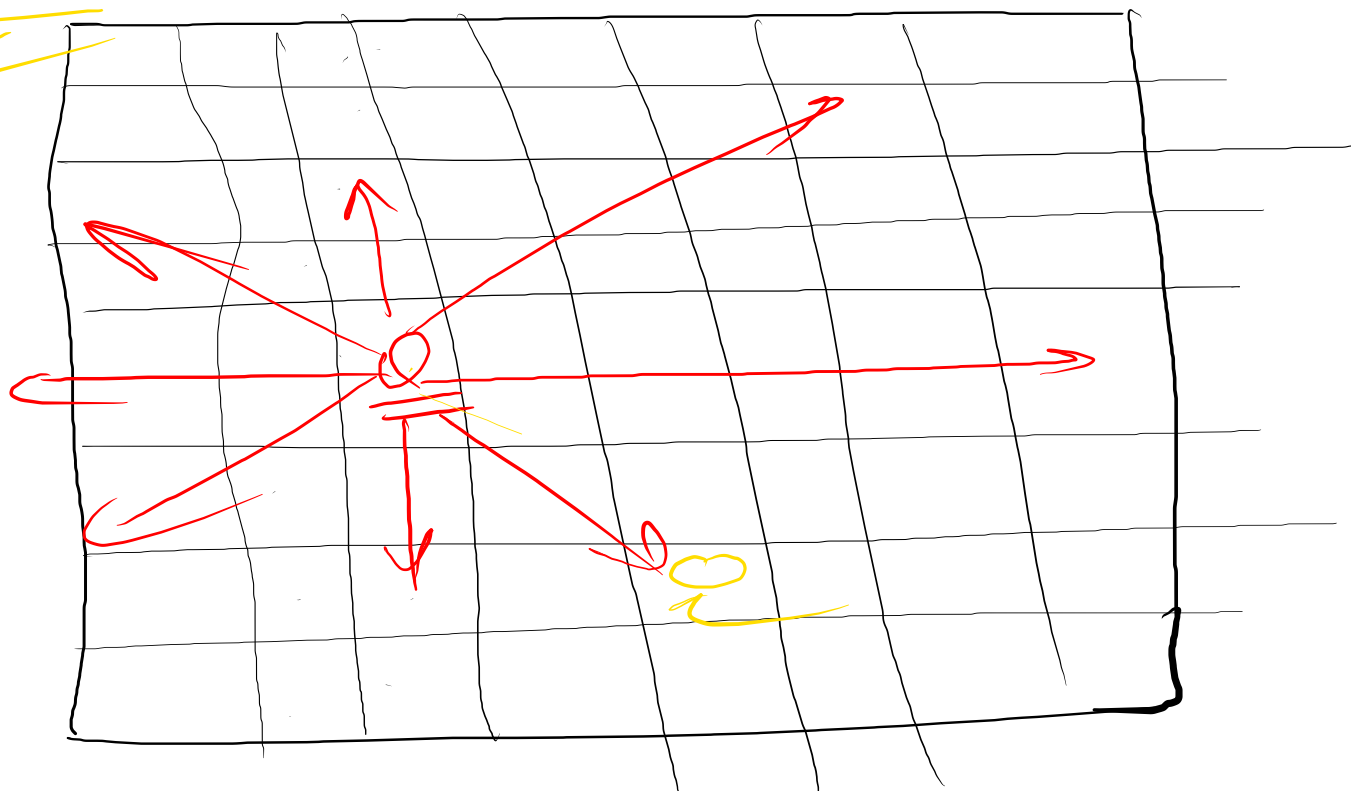


8x8 Matrix

Queen



4 queries

without taking
each other

	c_1	c_2	c_3	c_4
r_1	Q	x	x	x
r_2	x	x	x	Q
r_3	x	Q	x	x
r_4	x	x	x	x

4 Queens

Q

X



↙

x	x	Q	
Q	x	x	x
x	x	x	Q
x	Q	x	x

↘ (X4)

Hand

Backtracking

R ₁		x	Q	x
R ₂	Q	x		x
R ₃				Q
R ₄		Q		x

1 Row or 1 column

1 Queen

Recursion
+ BT

```
# Should return True if we are able to place all the queens
```

```
def solveNQutil(board, col, n):
```

```
    if(col >= n): #Base condition
```

```
        return True # Means we have been able to put queens in all the columns
```

```
    # Check for all the rows
```

```
    for row in range(n):
```

```
        if isSafeToPlaceQueen(board, row, col, n):
```

```
            board[row][col] = 1 #Set the queen
```

```
            # Recursively try for the next columns
```

```
            if solveNQutil(board, col+1, n):
```

```
                return True
```

```
            # Back Tracking
```

```
            board[row][col] = 0 # Queen can't be set here
```

```
    return False # Won't be able to place the queen
```

board

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

(board, 0, 4)

① col = 0, n = 4

↳ ② col = 1, n = 4

False

③ col = 2, n = 4

④ col = 2, n = 4

⑤ col = 3, n = 4

False

Backtracking



```
def isSafeToPlaceQueen(board , row , col , n):
```

```
# Check in the left side
```

```
for i in range(col):  
    if board[row][i]==1 :  
        return False
```

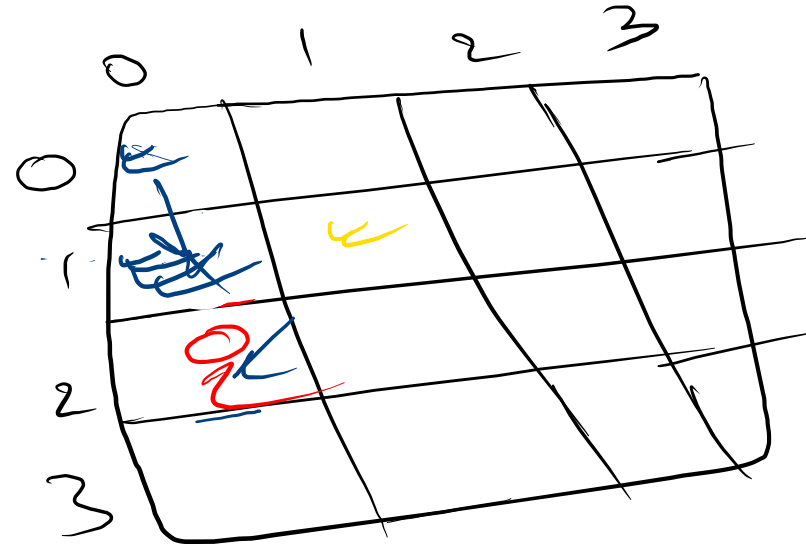
```
#Check in the upper left diagonal
```

```
for i, j in zip(range(row,-1,-1),range(col,-1,-1)):  
    if board[i][j] ==1:  
        return False
```

```
#Check in the lower left diagonal
```

```
for i,j in zip(range(row,n,1 ), range(col, -1,-1)):  
    if board[i][j]==1:  
        return False
```

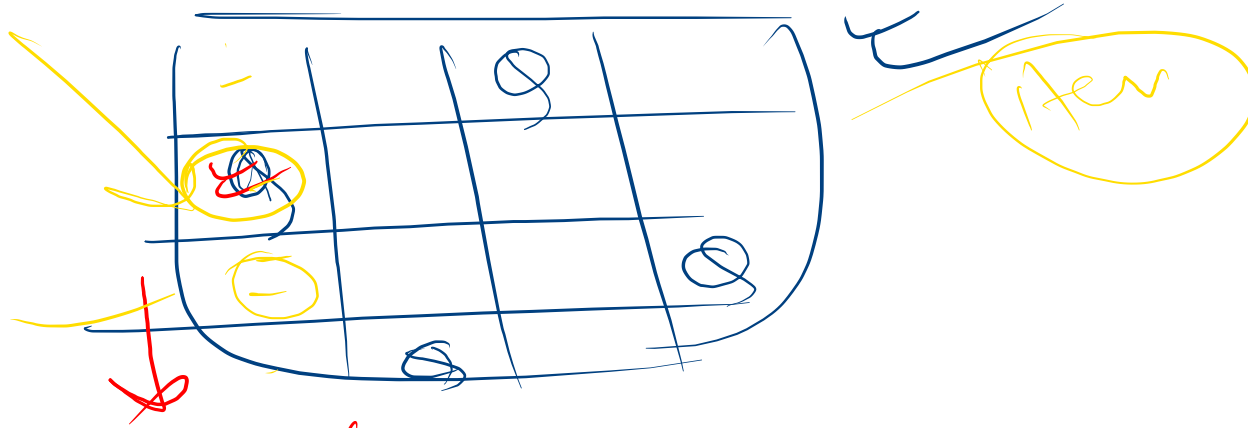
```
return True
```



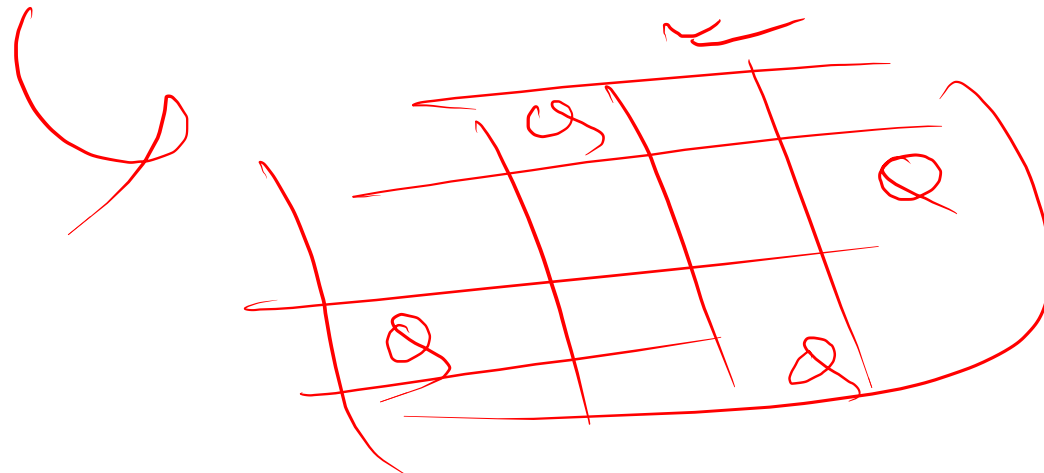
isSafeToPlace(board, row, col)

is a safe place

9:30 PM



Print all the
possibilities }



Array of words

word_break("leetcode", ["leet", "code"]) -> True

(leetcode)

fake

["I", "love", "sam", "sing", "boy"]

✓ sam sing

True

love boy

I love } words already
create

[¹I, "love", ²sam, "sing"]

↓
samsung

× samsung

sam sung

string of digits

Remainder

False

ksurf

① word = sameksurf

False

② word = (ksurf)

True

False

~~False~~

def wordBreak(wordList, word):

if word == "":

return True

else :

wordLen = len(word)

for i in range(1, wordLen+1):

if word[:i] in wordList and wordBreak(wordList, word[i:]):


return True

return False

[1, lane, ~~same~~, surf]

left → right

practice → 2 problems



Re-attempt all questions

$([3, 34, 4, 12, 5, 2], 9)$

✓ (2)

or

Remainder → (7)

$[3, 34, 4, 12, 5]$

$\left\{ \begin{array}{l} \text{Remainder (9)} \\ \hline [3, 34, 4, 12, 5] \end{array} \right\}$

Q9

[3, 34, 4, 12, 5, 2]

9

① select 2

Remain = 7

[3, 34, 4, 12, 5]

② Reject 2

Remain sum = 9

[3, 34, 4, 12, 5]

Reject 5

sum = 4

[3, 34, 4, 12]

select 5

sum = 4

[3, 34, 4, 12]

select 5

sum = 2

[3, 34, 4, 12]

Reject 5

sum = 7

[3, 34, 4, 12]

TC Questions } solution } (2 days) 10 questions

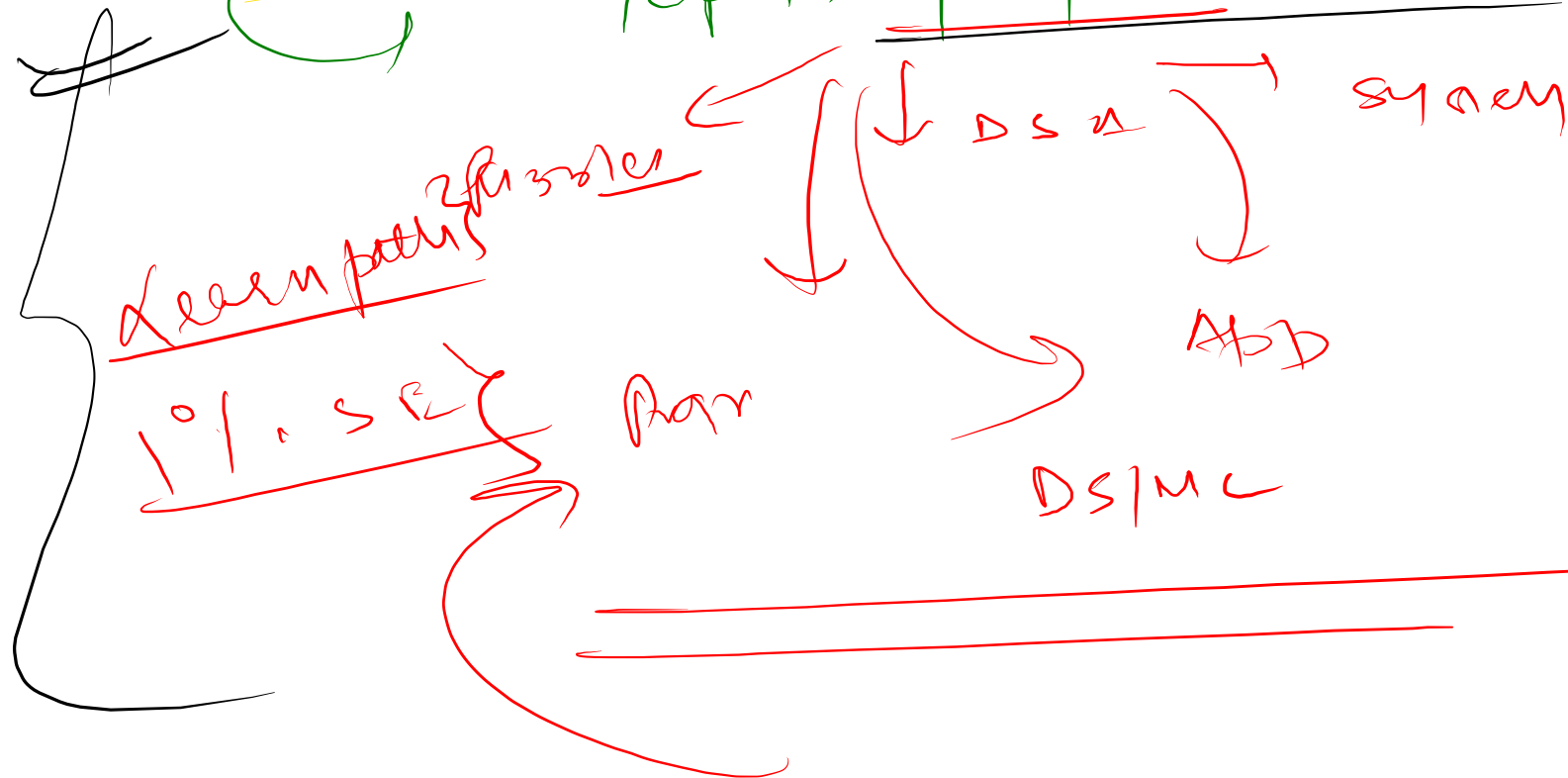
10 weeks Break
↓
100 questions

Community

24 → already

English

top 1% professional



2
Limited

Deeper

✓ CMS

Assignment
(
solutions)

