Abbreviation Using Backtracking

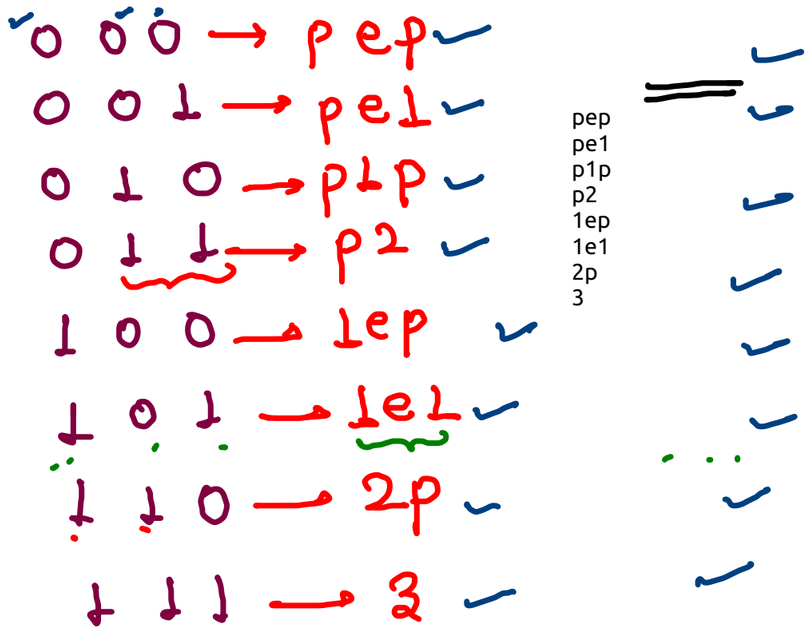
String Str: pep , length = 3 , Abbreviations 2^3

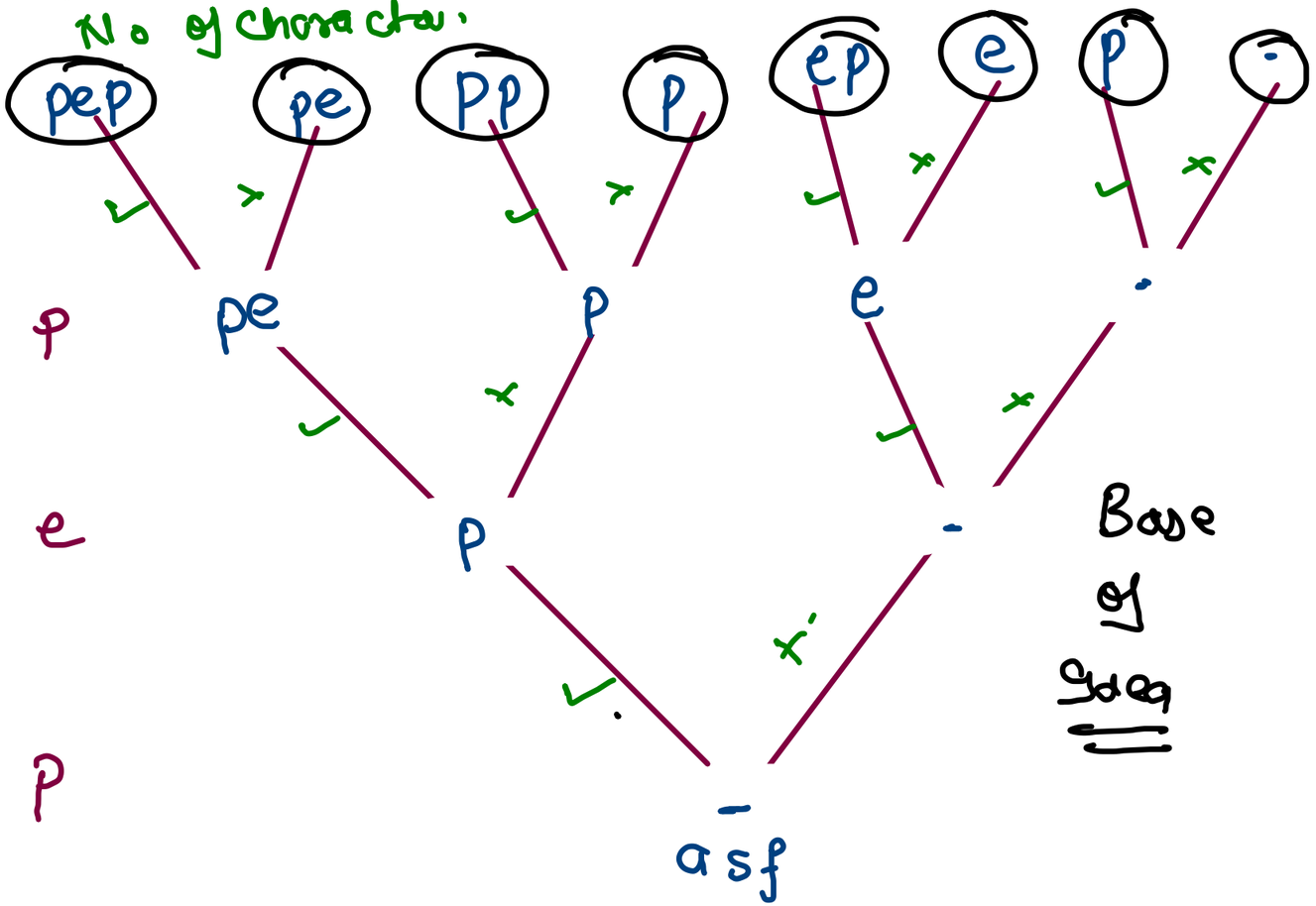
print all subsequence :→

level → character

option → Yes and No of character.

Numbers are add with No call





Base case :

String. {

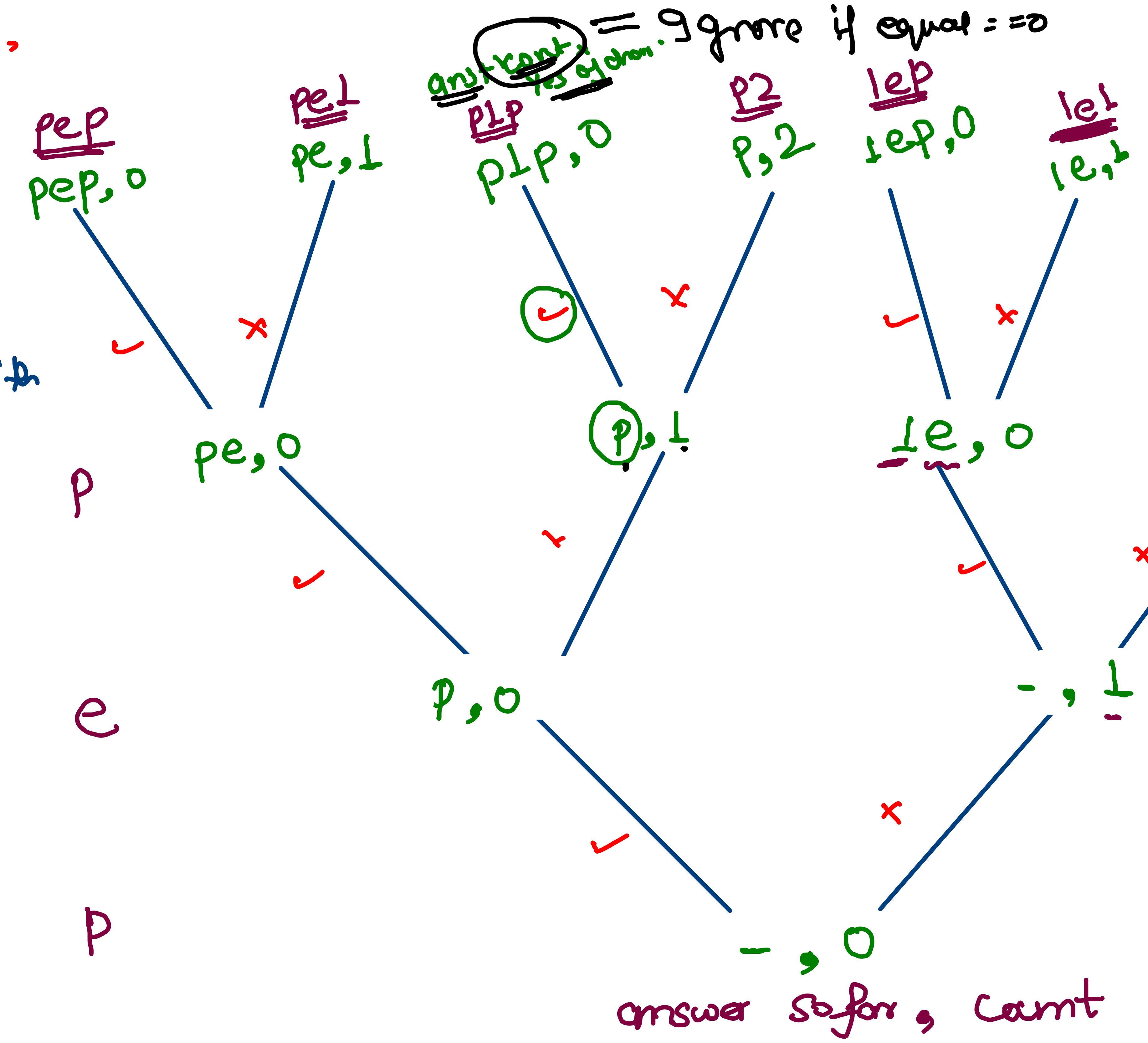
Base case

if(count != 0)
concatenate with answer so far

} else {

ignore

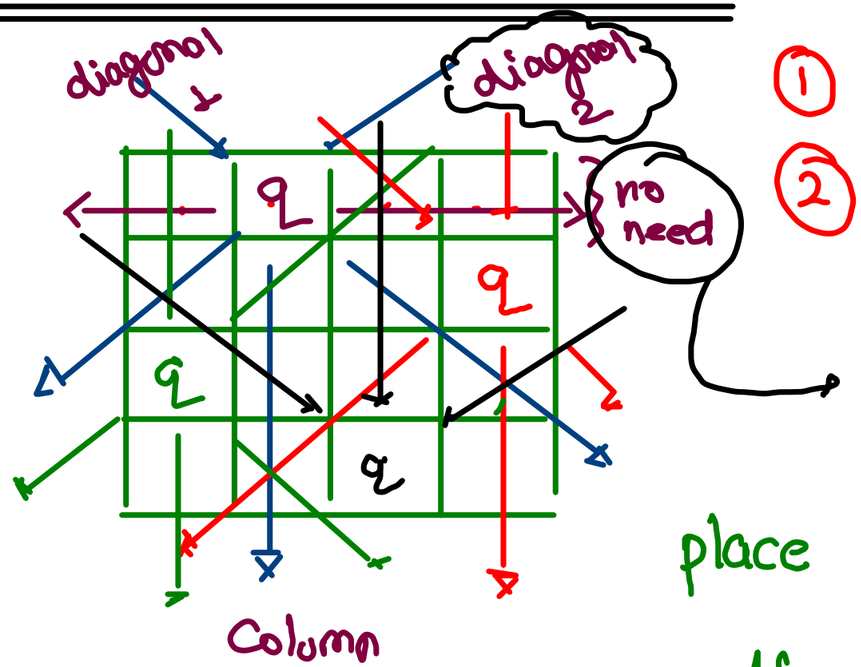
}



0	0	0	→	pep
0	0	1	→	pe1
0	1	0	→	p1p
0	1	1	→	p2
1	0	0	→	1ep
1	0	1	→	1e1
1	1	0	→	2p
1	1	1	→	3

pep

N Queens - Branch And Bound :



Is safe to place will reduce from O(4n) to O(1)

① In a row we have to place a queen certainly.

② In next row, we have to check if it is safe to place or not

Row will change on every level.

place queen 'q' and stop Columny and diagonals for future queen, this whole logic is called "branch and boand".

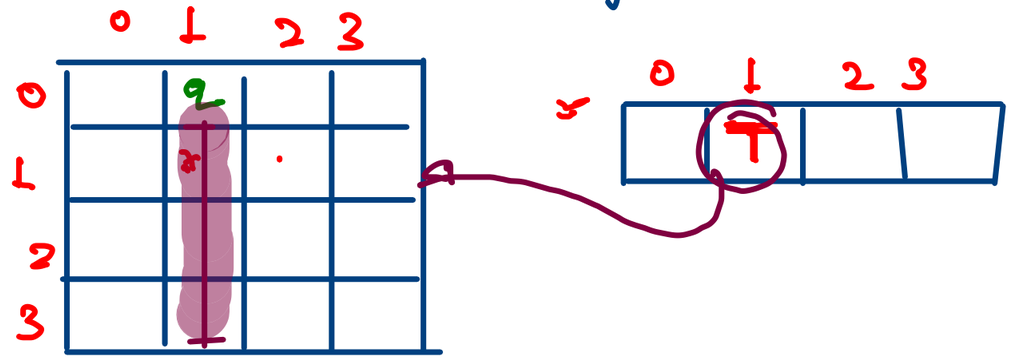
How to check isSafe in O(1) ??

To understand direction

diagonal 2 → normal diagonal (from top-right to Bottom left)

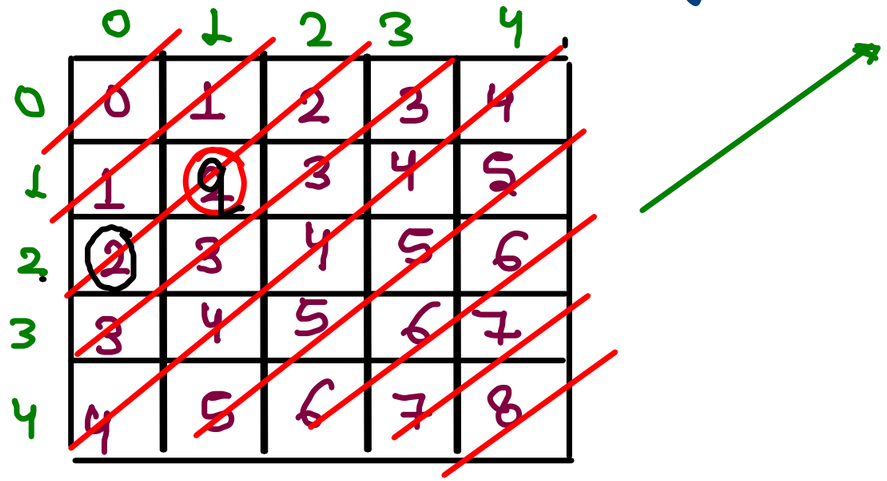
diagonal 1 → Reverse diagonal (from top-left to Bottom right)

① How to Ensure Safety with Column ?



boolean[] col = new boolean[column length]

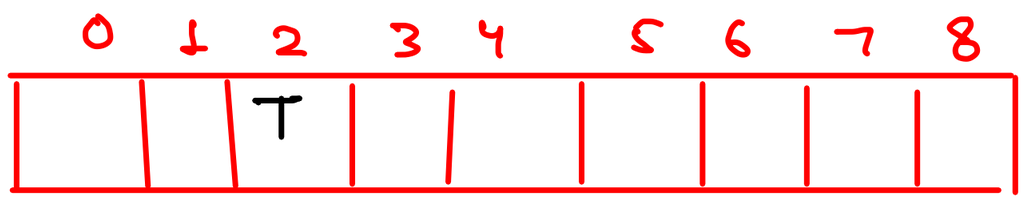
② How to ensure Safety of Normal diagonals (from top-right to Bottom left)



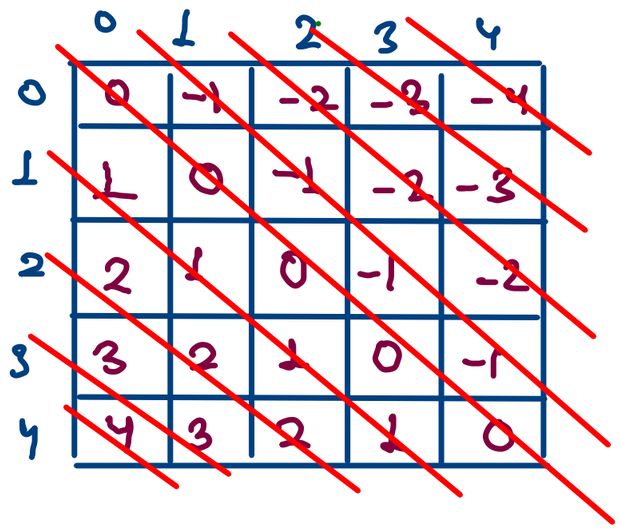
row + col

length of diagonals array to check safety is

2 * length - 1 ⇒ 2×5-1 = 9



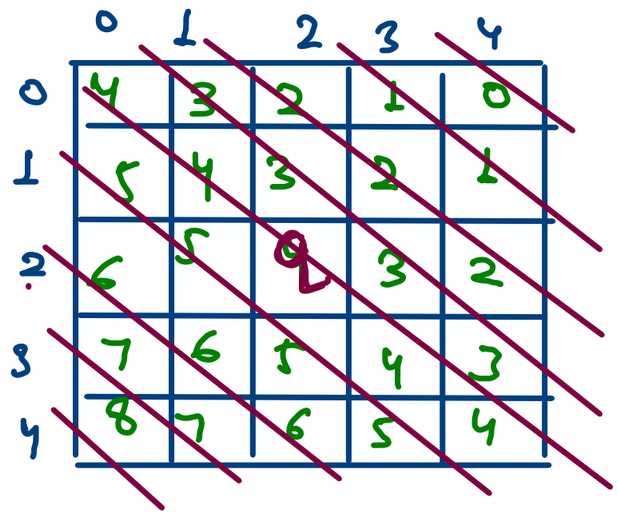
③ How to ensure safety in reverse diagonals (From top left to Bottom Right)



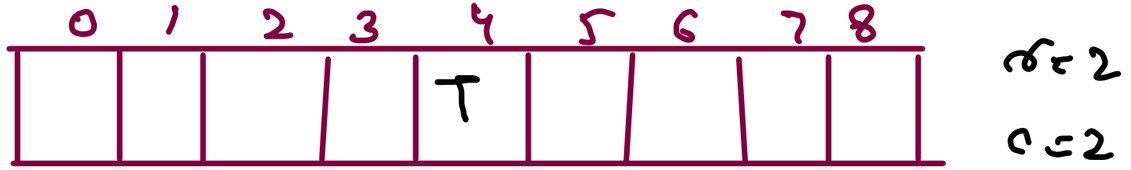
row - col

Now for proper Index

(row - col) + (length-1)



array for reverse diag.



(r - c + length-1)

= 2 - 2 + 5 - 1

= 4

For Isvalid

		Size	Index
col → 1 bit	column →	n.	(c)
d1 → 8 bit	diagonal1 (Normal)	2n-1	(r+c)
d2 → 8 bit	diagonal 2 (Reverse)	2n-1	(r-c + length-1)

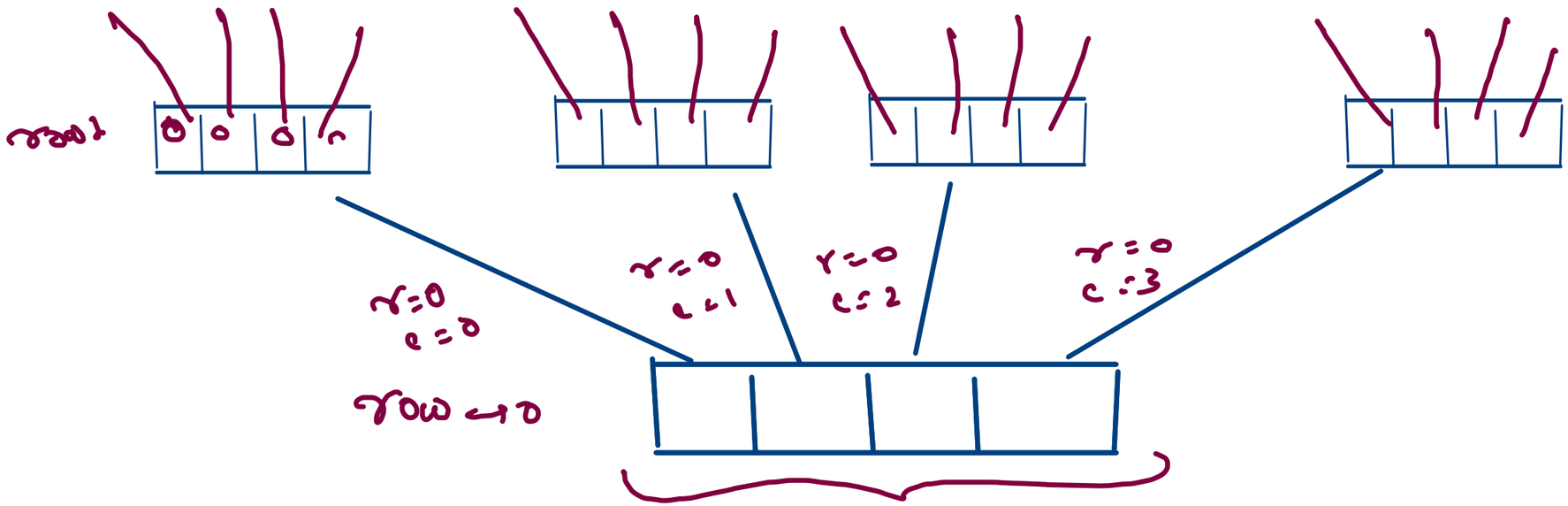
Array length = 2 * n - 1

O(3)

NOTE: We can't skip row.

Because we have n x n board and n queens.

row is on level, and we place 1 queen in row and chess all possibility with safety of queens.



Max Score :

Hint

level → words.

option → yes and No of word

yes → Check for freq. if available in Range then make yes call + also manage score with yess call

No → Every time.

4 → no. of words

dog cat dad good → 4 different words

9 → number

a b c d d d g o o → freq count

1 0 9 5 0 0 3 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0

a b c d e f g h _ _ _ _ o _ _ _ _ _ _ _ _ z

→ Score of Every character

Total possible sets = g^4

no. of words = 4.

① Generate set of words such that freq. of character will present in limits.

② Example

set 1 → dog dad → (21)

d → 3, o → 1, a → 1, g → 1

frequencies are in Range so this is valid set

set 2 → dog dad good

d → 4, o → 3, g → 2, a → 1

freq of d → 3 allowed, freq. of o → 2 allowed, that means this is not a valid set

③ From all possible valid set, what is max score??

["dog", "cat", "dad", "good"]

0	1	2	3	...	25
a	b	c	d		o
↓	↓	↓	↓		↓
1	1	1	3		2

a b c ddd g o

level → words.

option → Yes (If possible i.e. freq. available)

No

Given in Function

① - words array
- freq. array
→ Score array
- Indx

abcdd

abcddgo

dad

abcdd

abcdddgo

cat

abcdddgo

dog.

dog

Score.

1095003000000020000000000

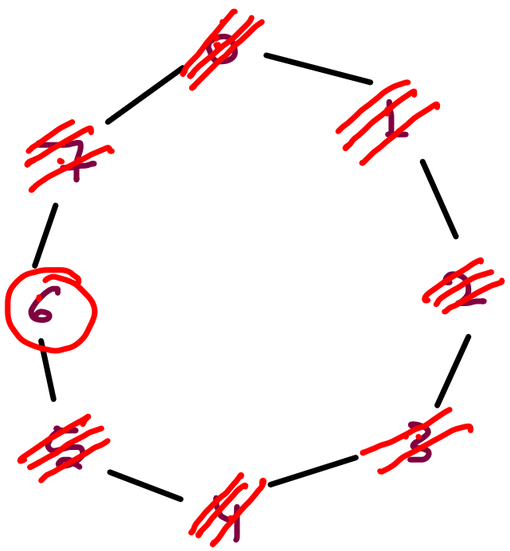
a b c d e f g h _ _ _ _ b _ _ _ _ _ _ _ _ z

max Return

Josephus Problem :

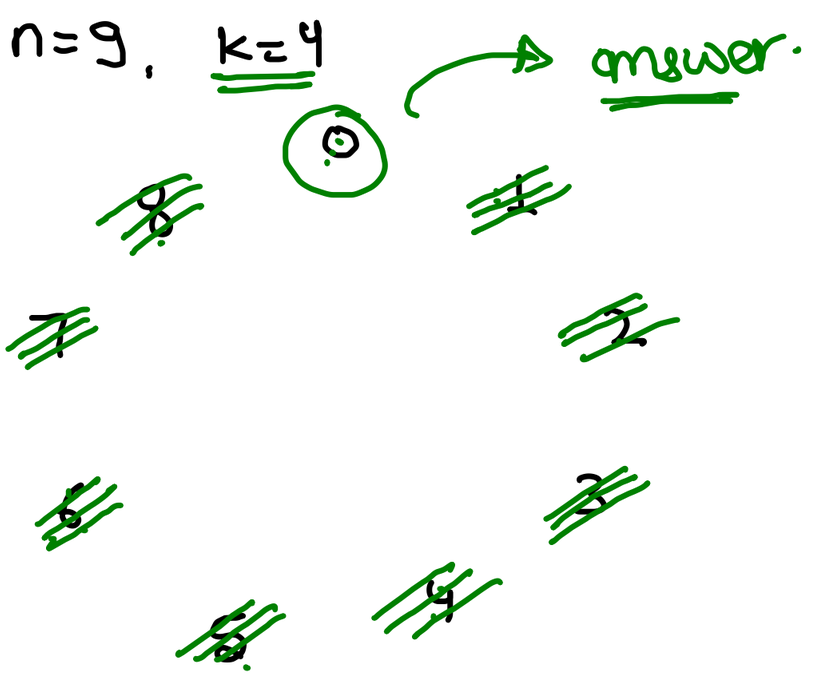
n → people , k → number

n = 8 , k = 3



answer = 6

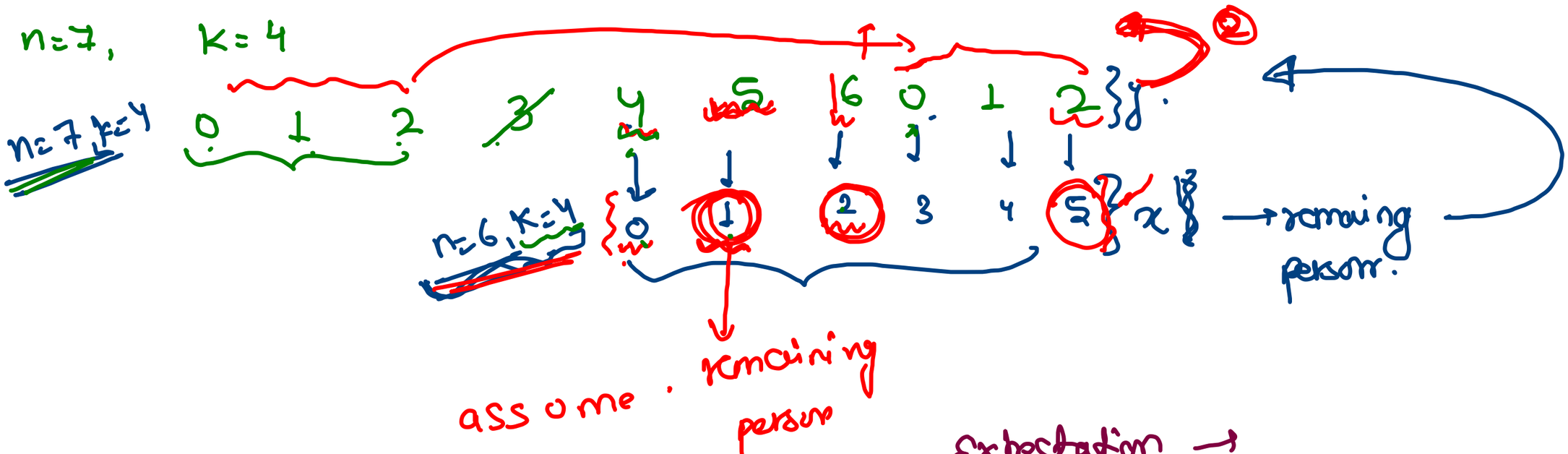
n = 9 , k = 4



answer.

n = 7 , k = 4.

answer



$y = (x + k)\%$ n.current_level

$(2+4)\%7$ $6\%7 = 6$

$(5+4)\%7$ $9\%7 = 2$

Expectation →

$(n, k) \longrightarrow$ remain person.

faith,

$(n-1, k) \longrightarrow$ remaining person

Merging → $(n-1, k) \longrightarrow$ faith.

return mapping pos

$(res + k) \% n$

n=2, k=4.

n=1

0

n=1, k=4

0 0

x=0

y = (x+k)%n = (0+4)%2 = 0

0 ~~0~~

n=2, k=4

0 x=0

y = (0+4)%3 = 1

~~0~~ 1 2

n=3, k=4

1

0 1 2

x=1

y = (1+4)%4 = 1

0 1 2 ~~3~~ 0 1 2

n=4, k=4

x=1

0 1 2 3

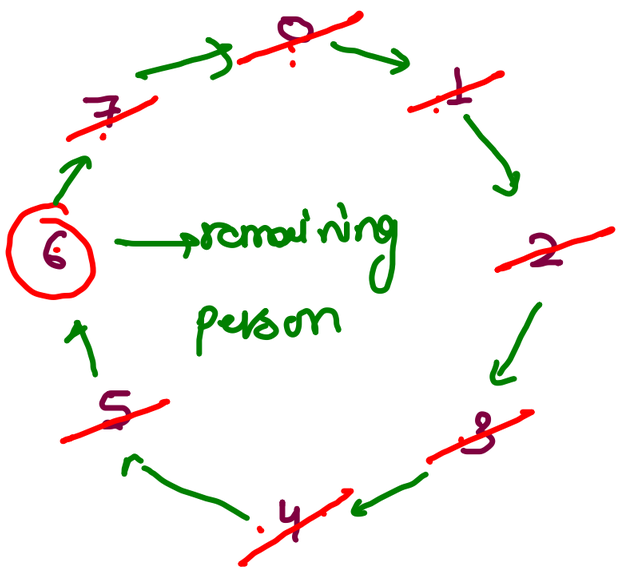
y = (1+4)%5 = 0

0 1 2 ~~3~~ 4 0 1 2

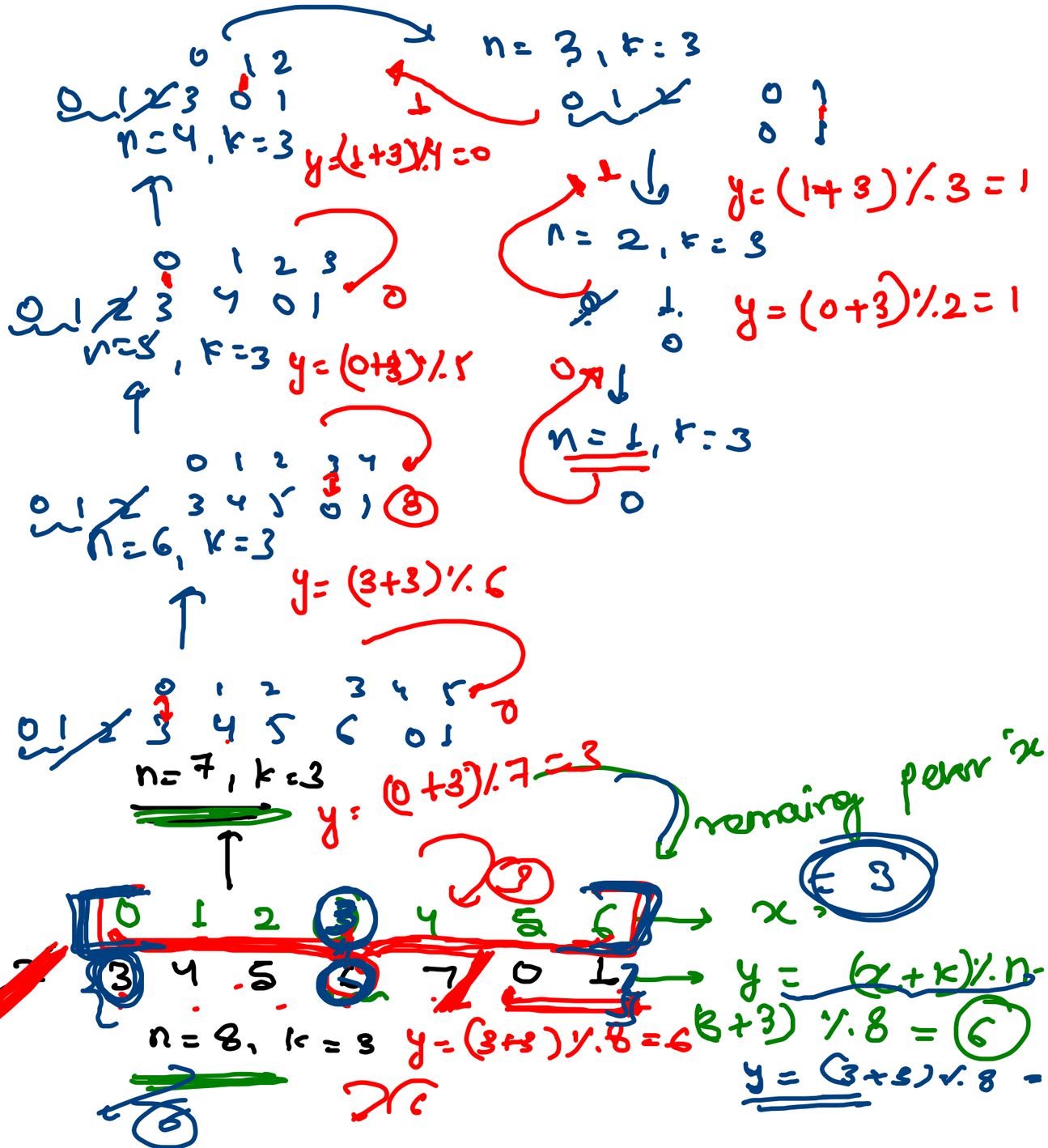
n=5, k=4

0

n = 8 k = 3



Linked list] Implementation is difficult but possible.



lexicographical printing.

n = 1000

1 to 1000] print all number in lexicographical order.

dictionary order

a → 1 →
b → 2 →
c → 3 →
d → 4 →
da → 41 →
db → 42 →
dc → 43 →
dca →
dcb →
⋮

1
10
100
1000
101
102
103
104
105
106
107
108
109

11
110
111
112
113

12
120
121
122
123

13 - - - 19
130
131
132
133

190
191
192
193

2
20
200
201
202

21
210
211

⋮