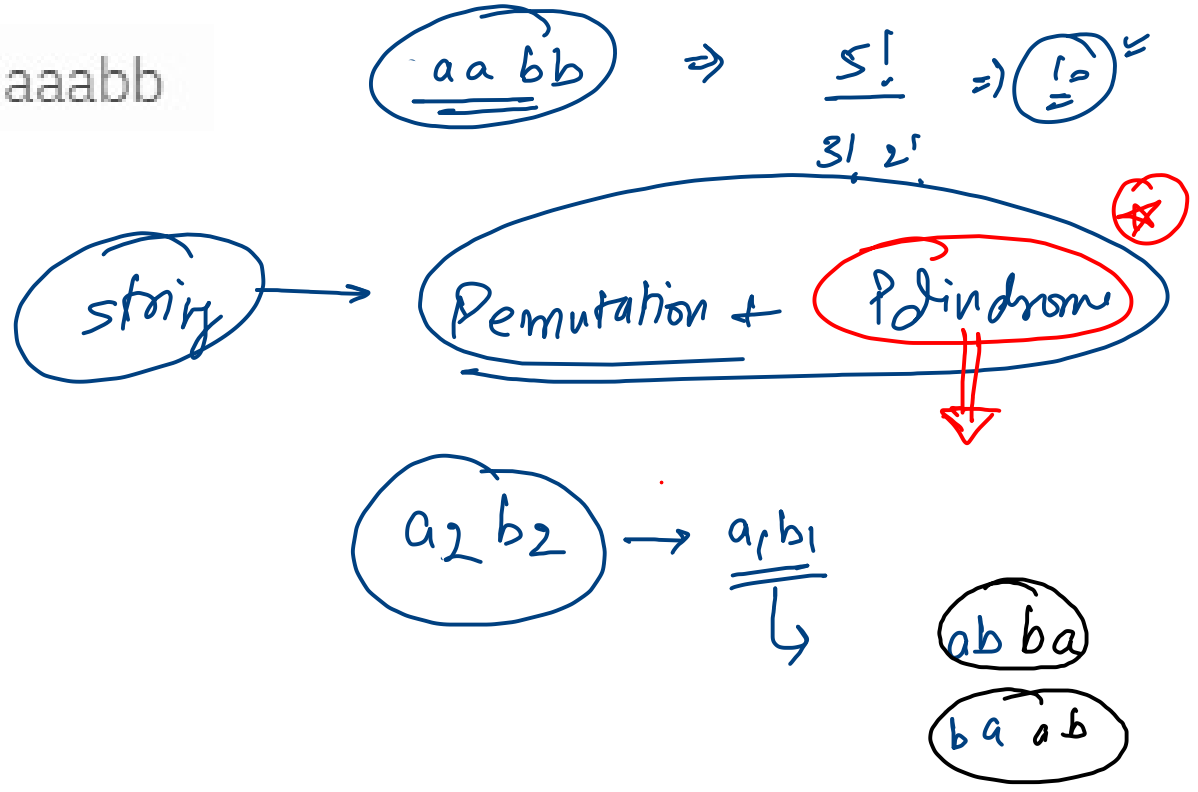
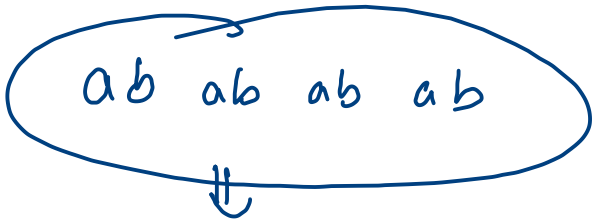
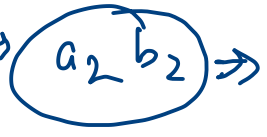
aaabb


aa bb
⇒
5!
⇒ 10
5 min
⇒ ?
3! 2!
string
Permutation +
Palindrom
a2 b2
a1 b1
ab ba
ba ab



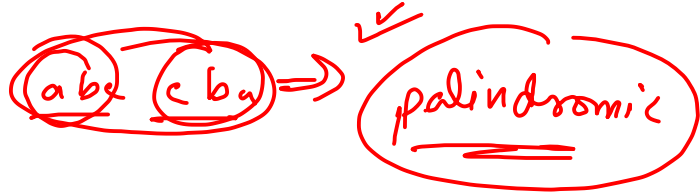
$a_4 b_4$

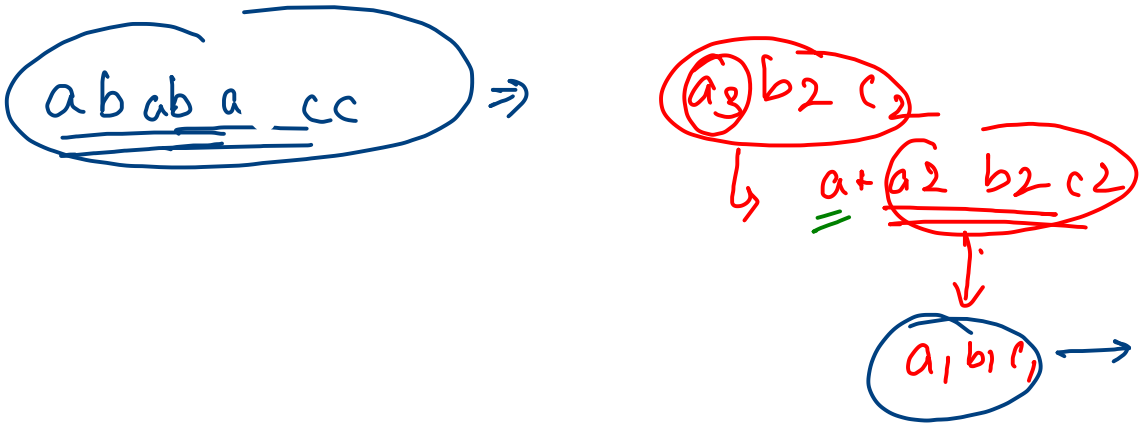


Permute

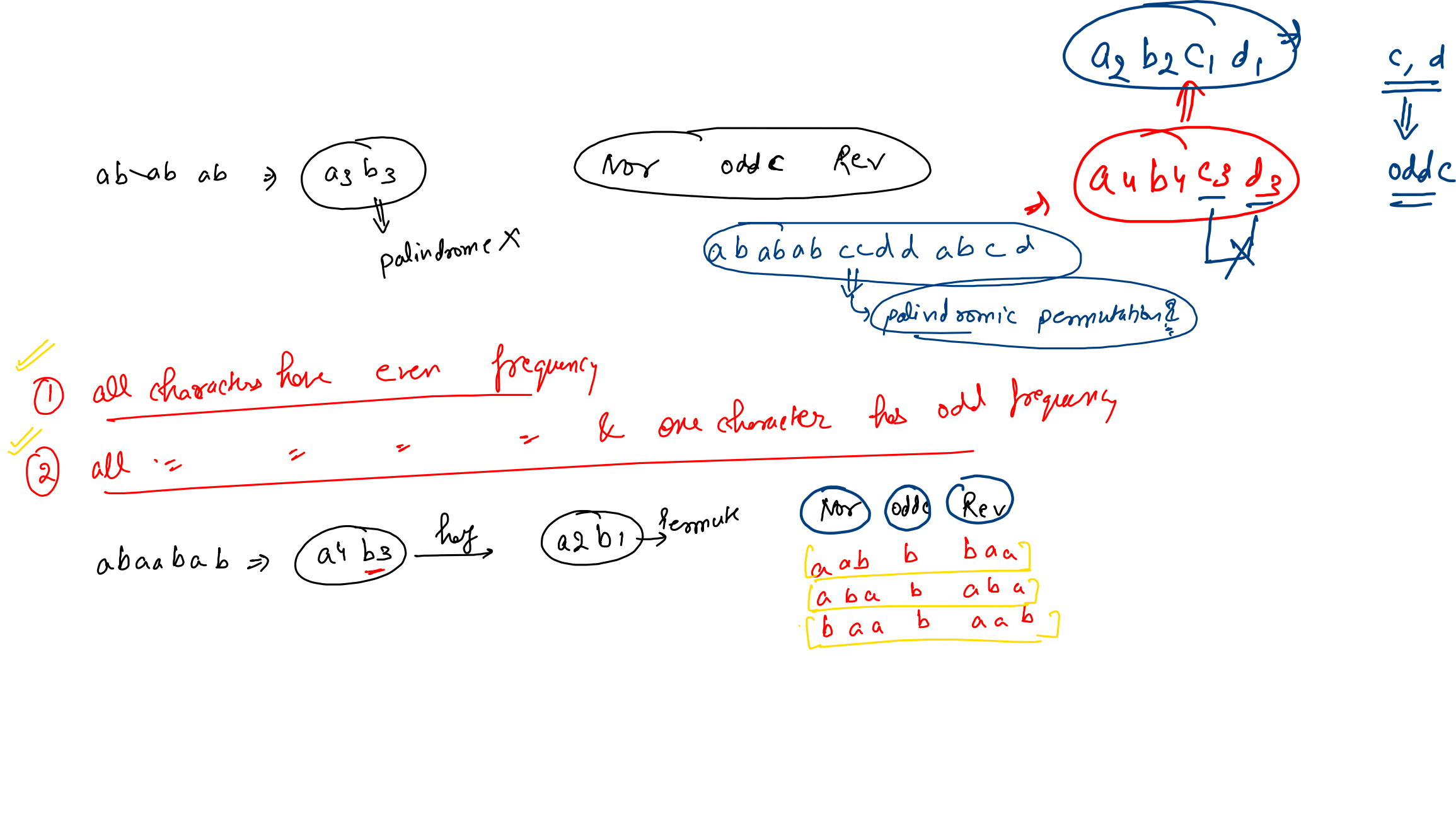
Nor ✓ Rev

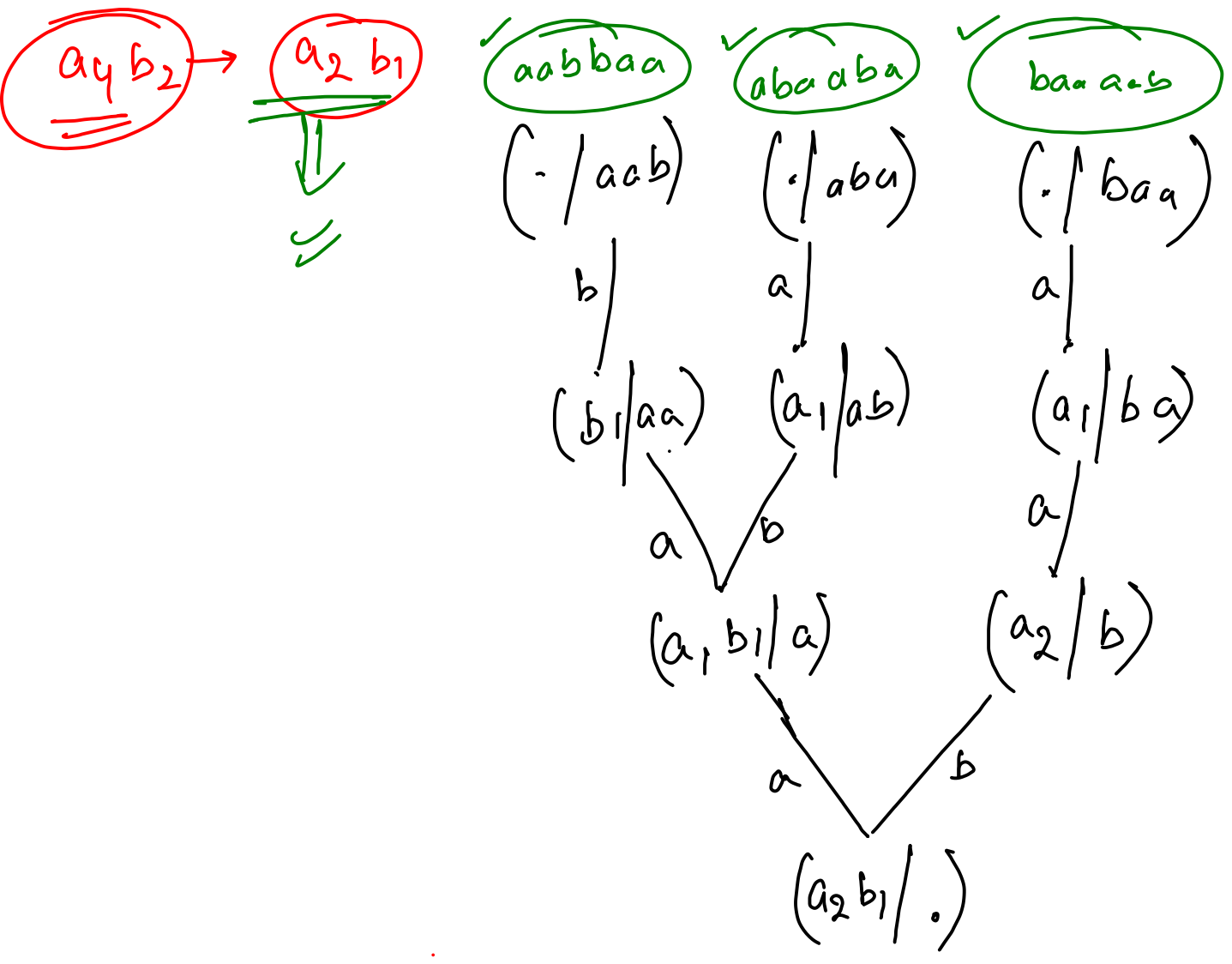
aabb	bbaa
abab	baba
abba	abba
bbaa	aabb
baba	abab
baab	baab



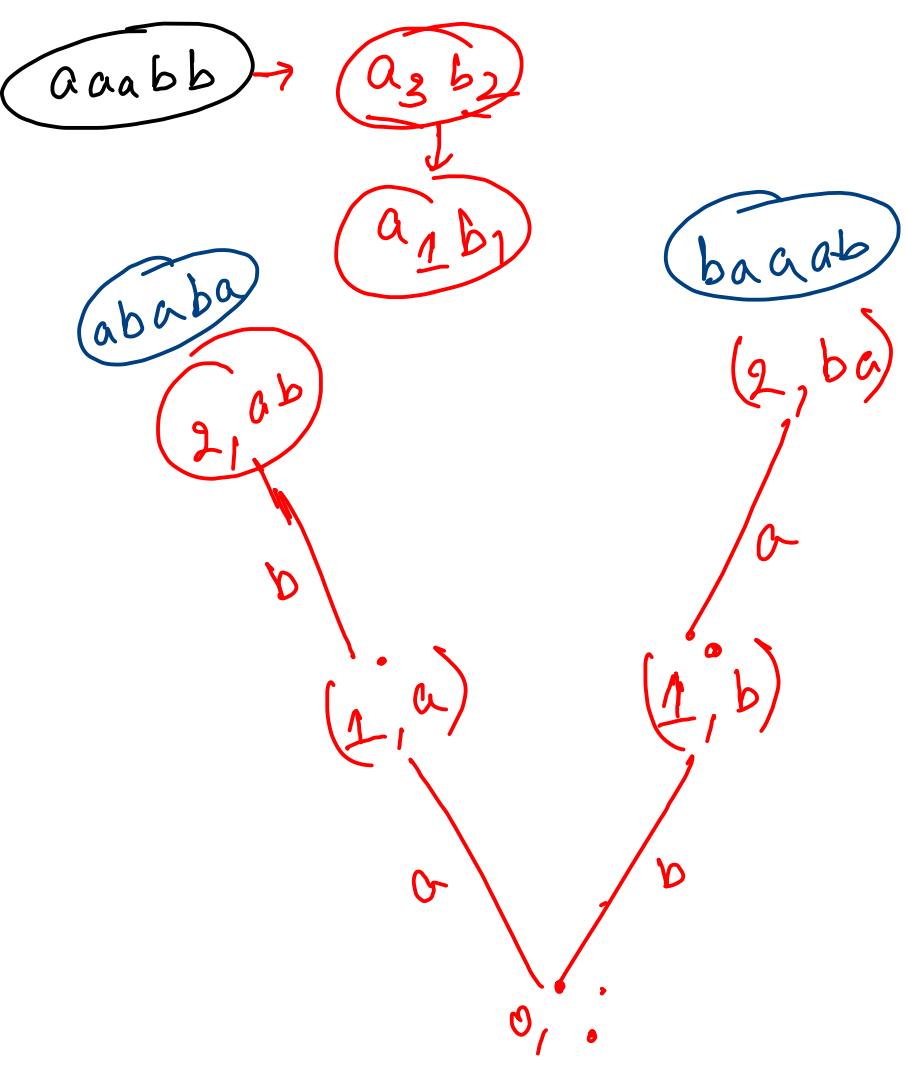


Permute	odd c ⇓	Rev
abc	a	cba ⇒ abcacba
bac	a	cab ⇒ bacacab
acb	a	bca = acb abca
bca	a	acb ⇒ bca acacb
cab	a	bac ⇒ cab abac
cba	a	abc ⇒ cba a abc

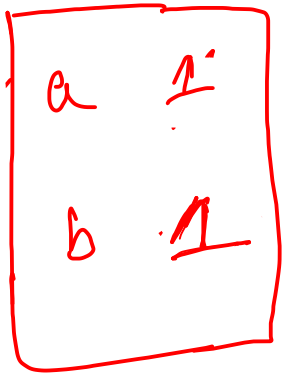

ab ab ab ⇒ a3 b3
palindrome ✗
Nor odd Rev
a2 b2 c1 d1
a4 b4 c3 d3
c, d
odd
a b ab ab c d d ab c d
palindromic permutation ?
1 all characters have even frequency
2 all = = = = & one character has odd frequency
abaabab ⇒ a4 b3
half
a2 b1
permute
Nor
odd
Rev
a ab b baa
aba b aba
baa b aab




```java
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    String str = scn.next();
    HashMap<Character, Integer> fmap = new HashMap<>();
    for (int i = 0; i < str.length(); i++) {
        char ch = str.charAt(i);
        fmap.put(ch, fmap.getOrDefault(ch, 0) + 1);
    }

    Character oddc = null;
    int count = 0;
    int len = 0;
    for(char ch : fmap.keySet()){
        int freq = fmap.get(ch);

        if(freq % 2 == 1){
            oddc = ch;
            count++;
        }
        len += (freq/2);
        fmap.put(ch,freq/2);
    }

    if(count > 1){
        System.out.println("-1");
    }else{
        generatepw(0,len,fmap,oddc,"");
    }
}
```



oddc = ~~null~~ a Count = ~~0~~ 1 len = 0+1+1 = 2 → ts




```java
public static String reverse(String str){
    StringBuilder sb = new StringBuilder(str);
    sb.reverse();
    return sb.toString();
}
public static void generatepw(int cs, int ts, HashMap<Char
    if(cs == ts){
        String rev = reverse(asf);
        if(oddc != null){
           System.out.println(asf+oddc+rev);
        }else{
           System.out.println(asf+rev);
        }
        return;
    }

    for(char ch :  fmap.keySet()){
        int freq = fmap.get(ch);
        if(freq > 0){
            fmap.put(ch,freq-1);
            generatepw(cs+1,ts,fmap,oddc,asf+ch);
            fmap.put(ch,freq);
        }
    }
}
```

K Partition

☆ Non-Empty

$n = 3$ $K = 2$

1, 2, 3

$n = 3$ $K = 5$ ✗

1, 2, 3

$n = 3$, $K = 3$ ⇒ 1

1, 2, 3

1 | 2 | 3

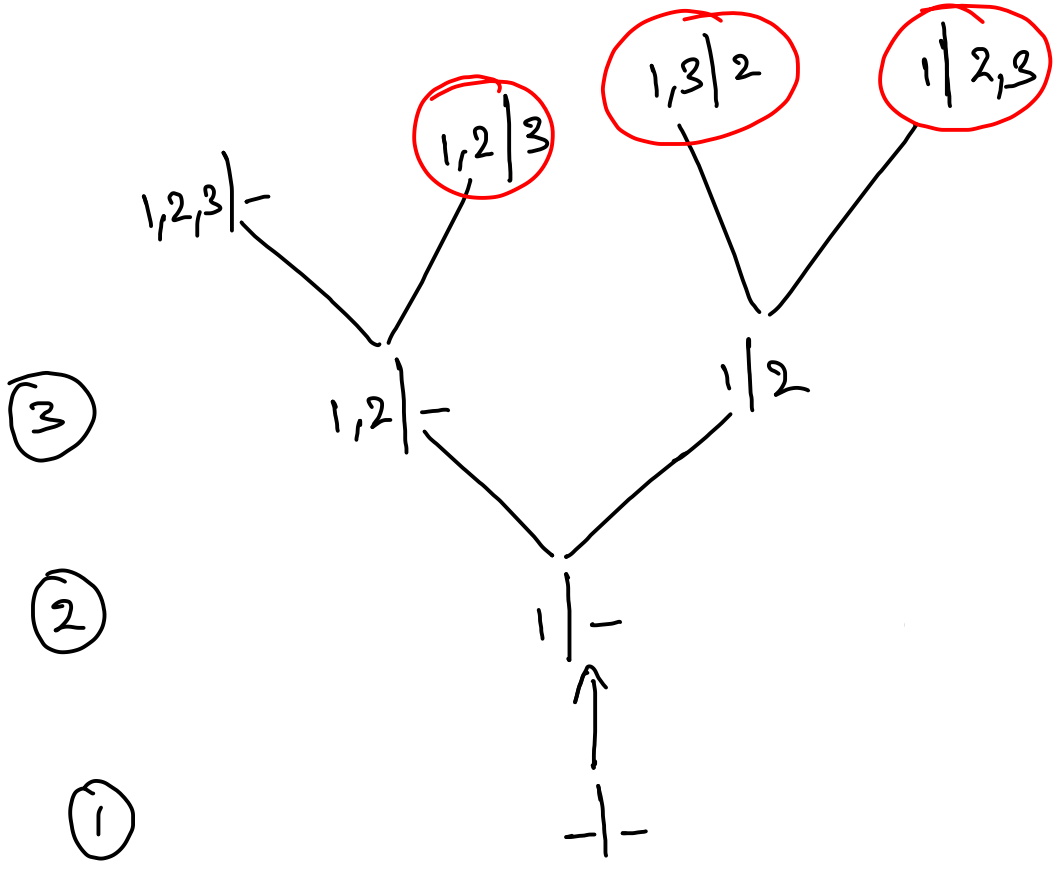
$n=3$, $k=2$

1, 2, 3

[1, 2] [3]

[1, 3] [2]

[1] [2, 3]



$n=4,\ k=3$

↳ 1,2,3,4

[1,2][3][4]

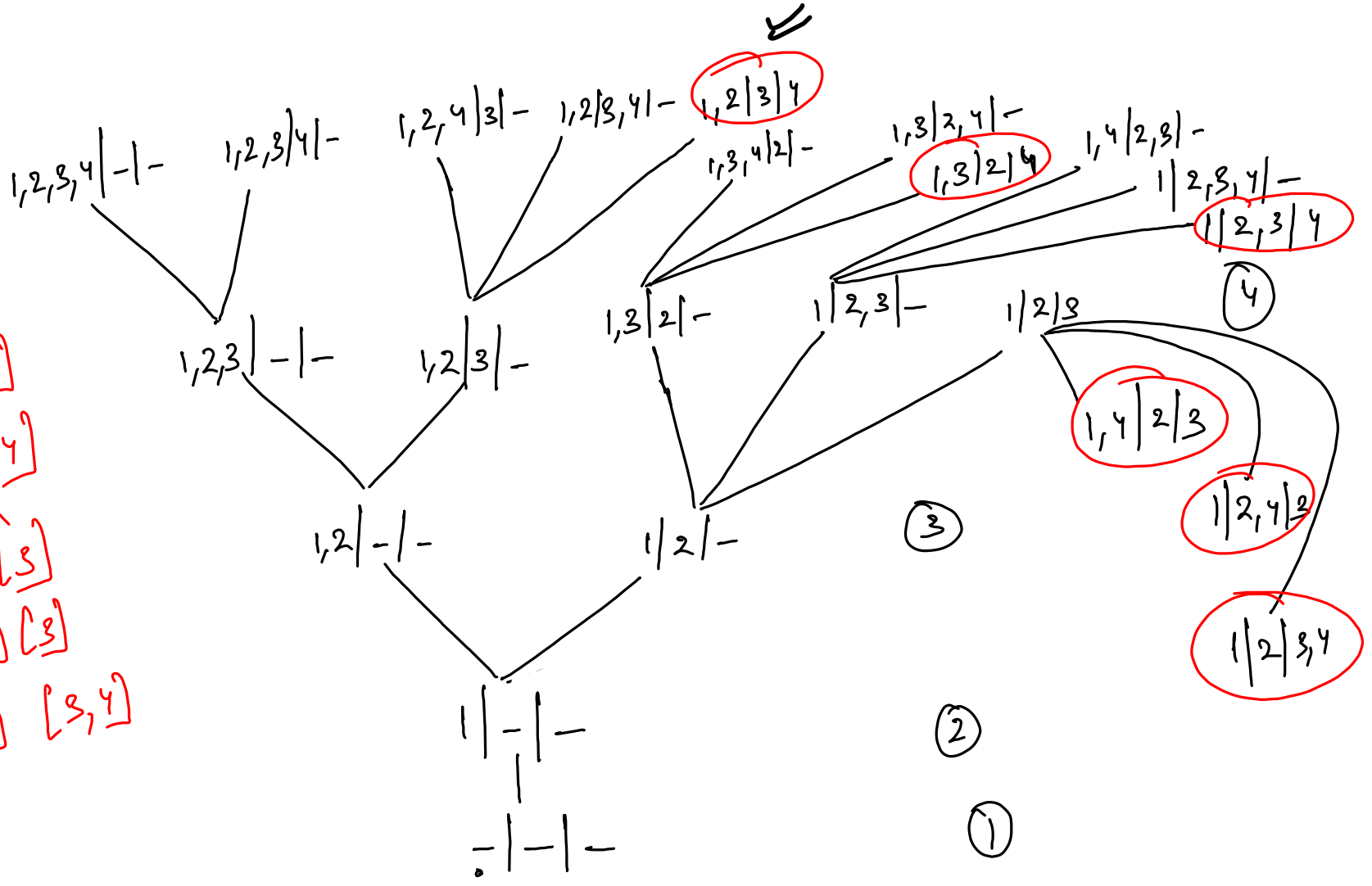
[1,3][2][4]

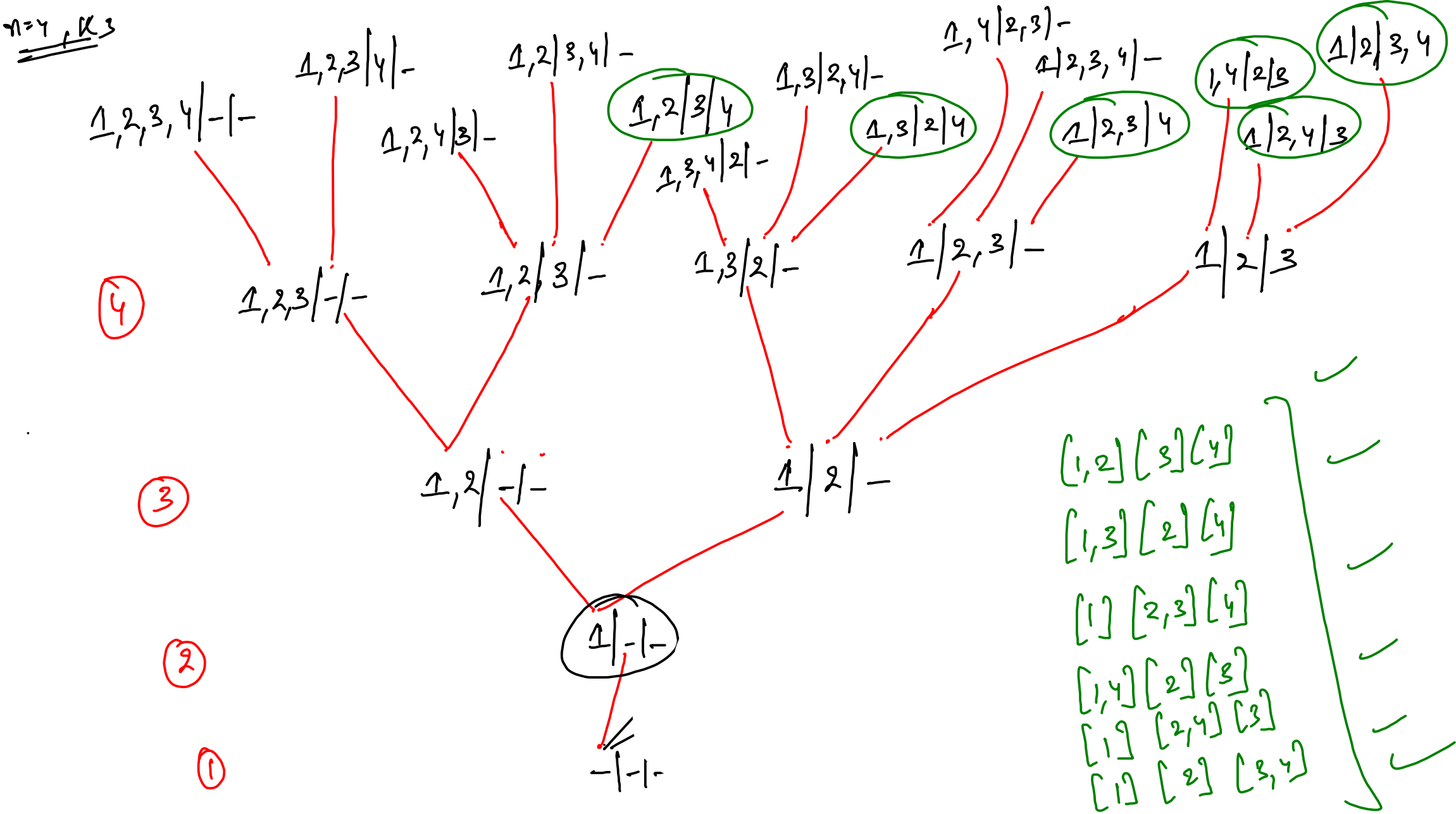
[1][2,3][4]

[1,4][2][3]

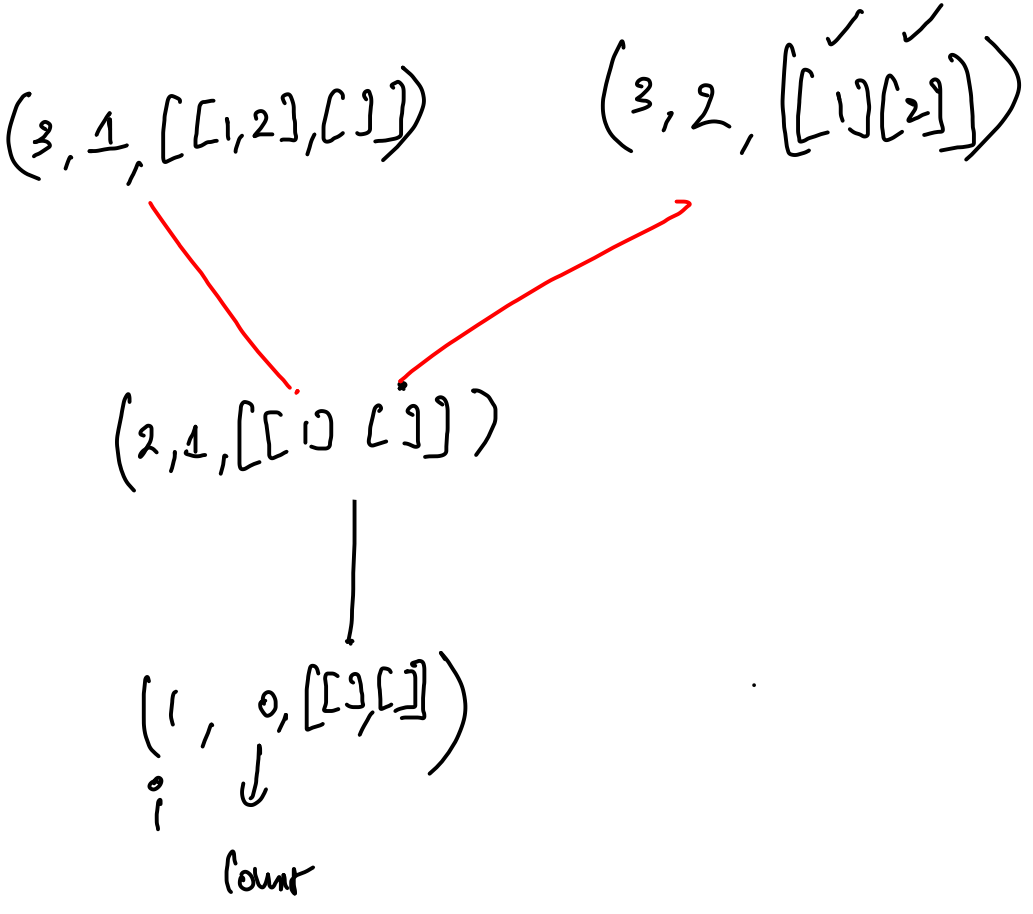
[1][2,4][3]

[1][2][3,4]

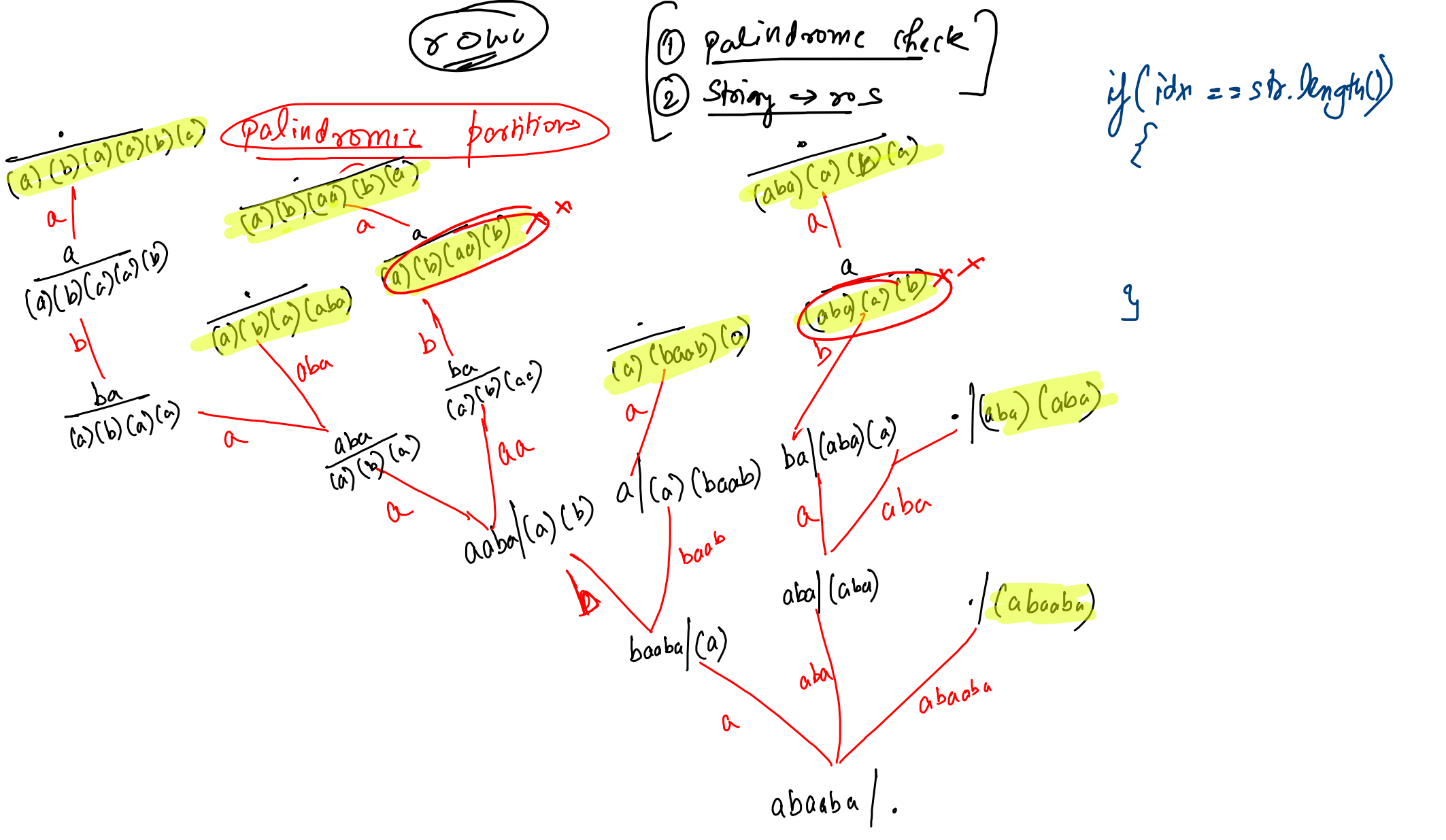


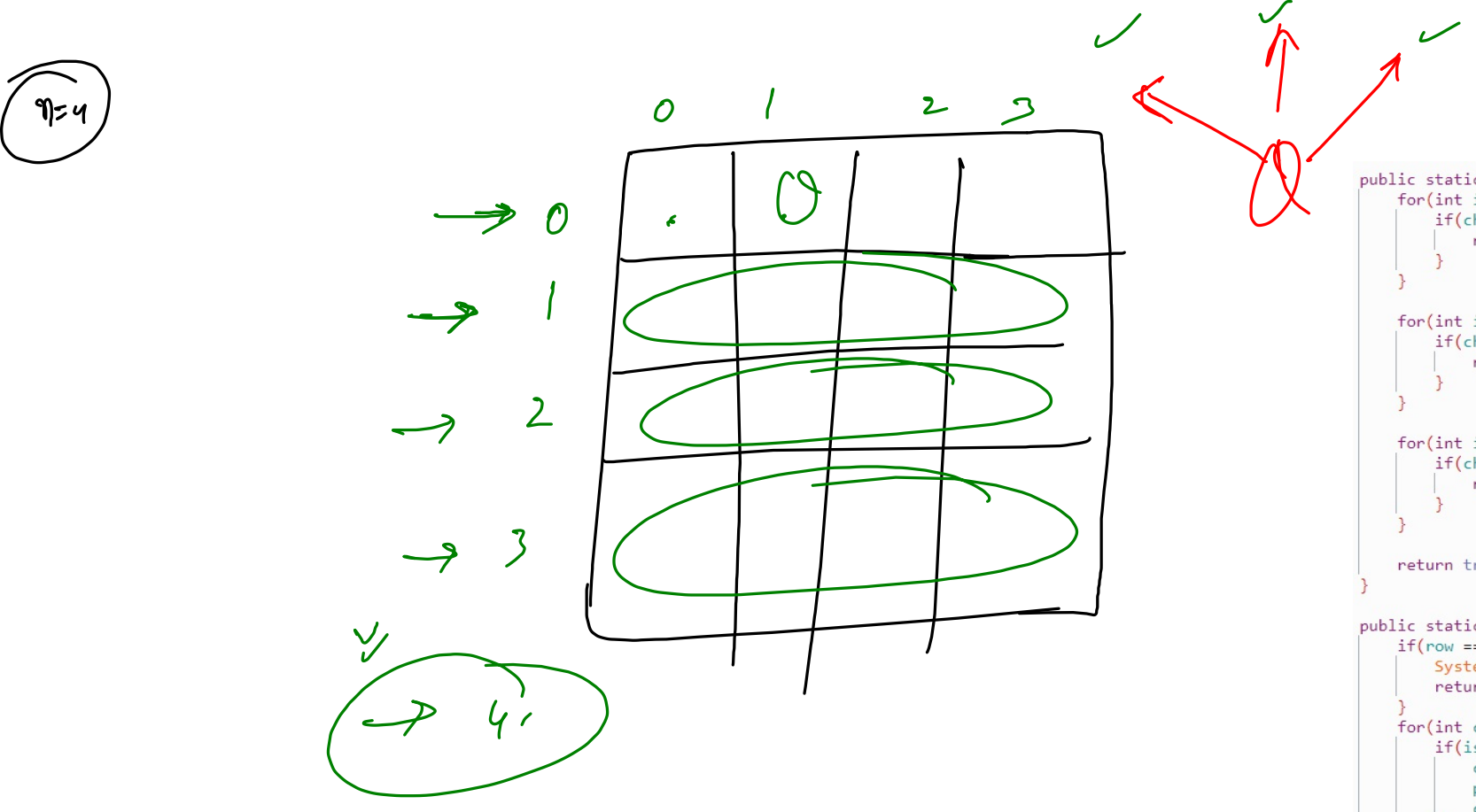

n=4, k=3
1,2,3,4|-|-
1,2,3|4|-
1,2,4|3|-
1,2|3,4|-
1,2|3|4
1,3,4|2|-
1,3|2,4|-
1,3|2|4
1,4|2,3|-
1|2,3,4|-
1|2,3|4
1,4|2|3
1|2,4|3
1|2|3,4
4
1,2,3|-|-
1,2|3|-
1,3|2|-
1|2,3|-
1|2|3
3
1,2|-|-
1|2|-
2
1|-|-
1
-|-|-
[1,2] [3] [4]
[1,3] [2] [4]
[1] [2,3] [4]
[1,4] [2] [3]
[1] [2,4] [3]
[1] [2] [3,4]

n=3 | k=2



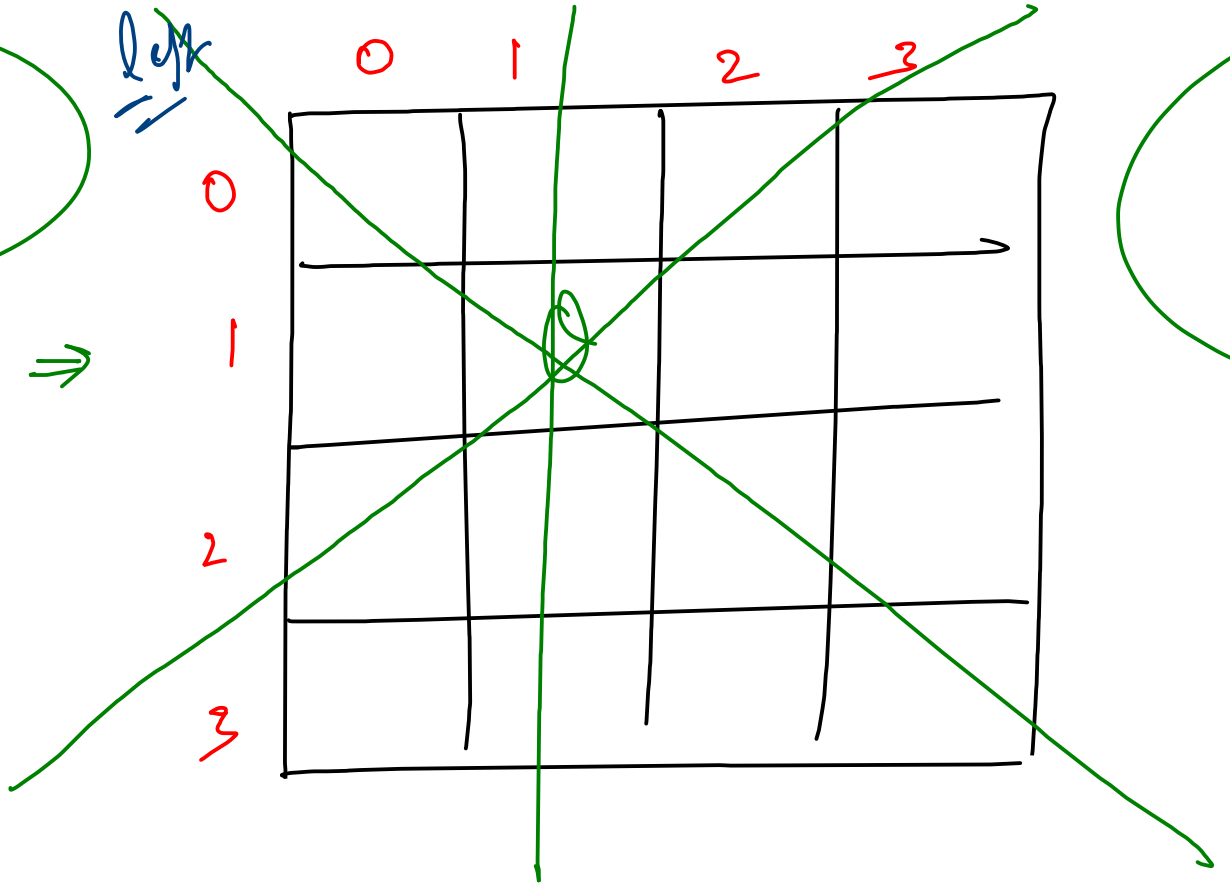

```java
static int count;
public static void solution(int i, int n, int k,
    if(i > n){
        if(csne == k){
            count++;
            System.out.print(count+". ");
            for(ArrayList<Integer> set : ans){
                System.out.print(set+" ");
            }
            System.out.println();
        }
        return;
    }

    for(ArrayList<Integer> set : ans){
        if(set.size() > 0){
            set.add(i);
            solution(i+1,n,k,csne,ans);
            set.remove(set.size()-1);
        }else{
            set.add(i);
            solution(i+1,n,k,csne+1,ans);
            set.remove(set.size()-1);
            break;
        }
    }
}
```


recursion
palindrome check
String → recursion
Palindromic partitions
if(idx == str.length())
{
}
(a)(b)(a)(c)(b)(c)
a
a
(a)(b)(c)(a)(b)
b
ba
(a)(b)(a)(c)
a
(a)(b)(a)(aba)
aba
(a)(b)(a)
a
(a)(b)(aa)(b)(a)
a
a
(a)(b)(aa)(b)
b
ba
(a)(b)(aa)
aa
aaba|(a)(b)
b
(a)(baab)(a)
a
a|(a)(baab)
baab
baaba|(a)
a
(aba)(a)(b)(a)
a
a
(aba)(a)(b)
b
ba|(aba)(a)
a
aba|(aba)
aba
abaaba|.
(aba)(aba)
aba
(abaaba)
abaaba|.

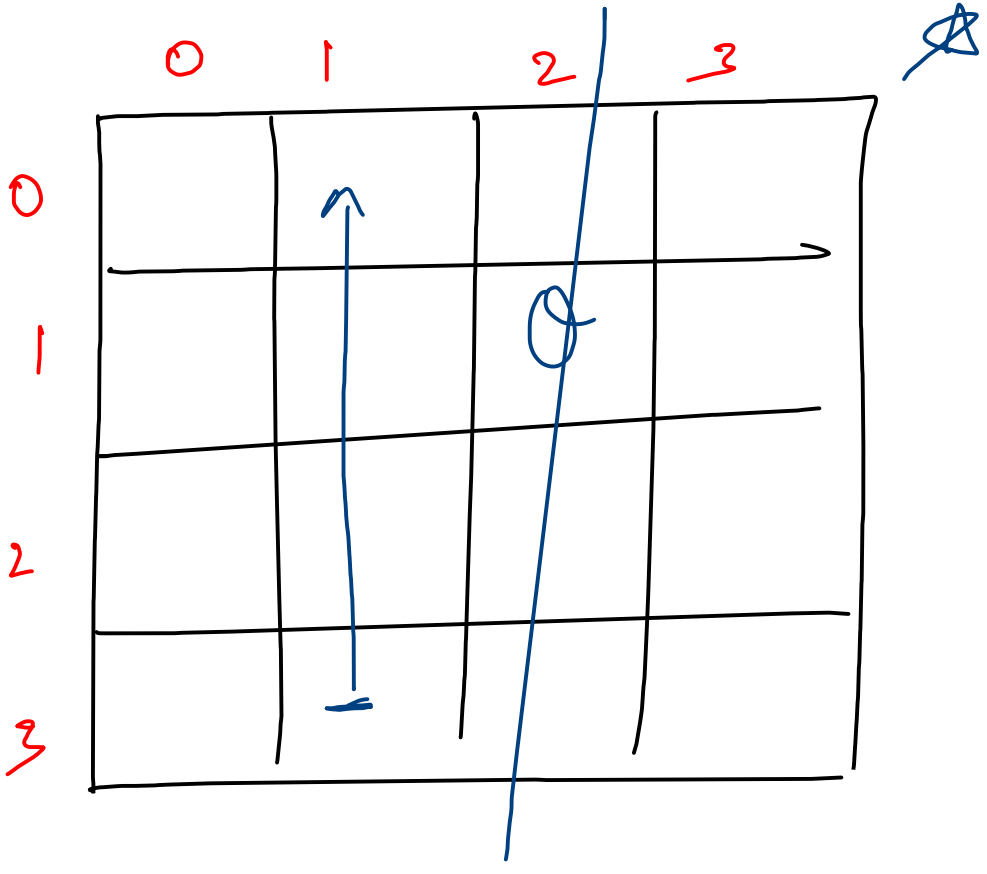



```java
public static boolean isSafe(int chess[][],int r,int c){
    for(int i = r-1 , j = c ; i >= 0 ; i--){
        if(chess[i][j] == 1){
            return false;
        }
    }

    for(int i = r-1 , j = c+1 ; i >= 0 && j < chess[0].length ; i--,j++){
        if(chess[i][j] == 1){
            return false;
        }
    }

    for(int i = r-1 , j = c-1 ; i >= 0 && j >= 0 ; i--,j--){
        if(chess[i][j] == 1){
            return false;
        }
    }

    return true;
}

public static void printNQueens(int[][] chess, String qsf, int row) {
    if(row == chess.length){
        System.out.println(qsf+".");
        return;
    }
    for(int col = 0 ; col < chess.length ; col++){
        if(isSafe(chess,row,col)){
            chess[row][col] = 1;
            printNQueens(chess,qsf+row+"-"+col+", ",row+1);
            chess[row][col] = 0;
        }
    }
}
```

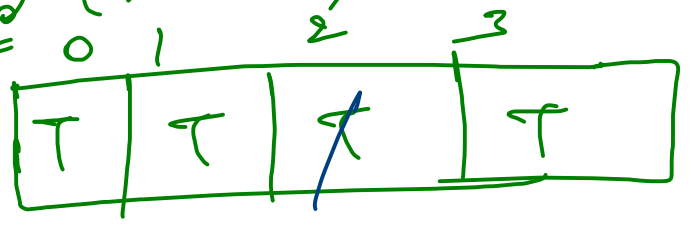
Time efficient
+
Extra Space



1. Col
2. Left Diag
3. Right Diag

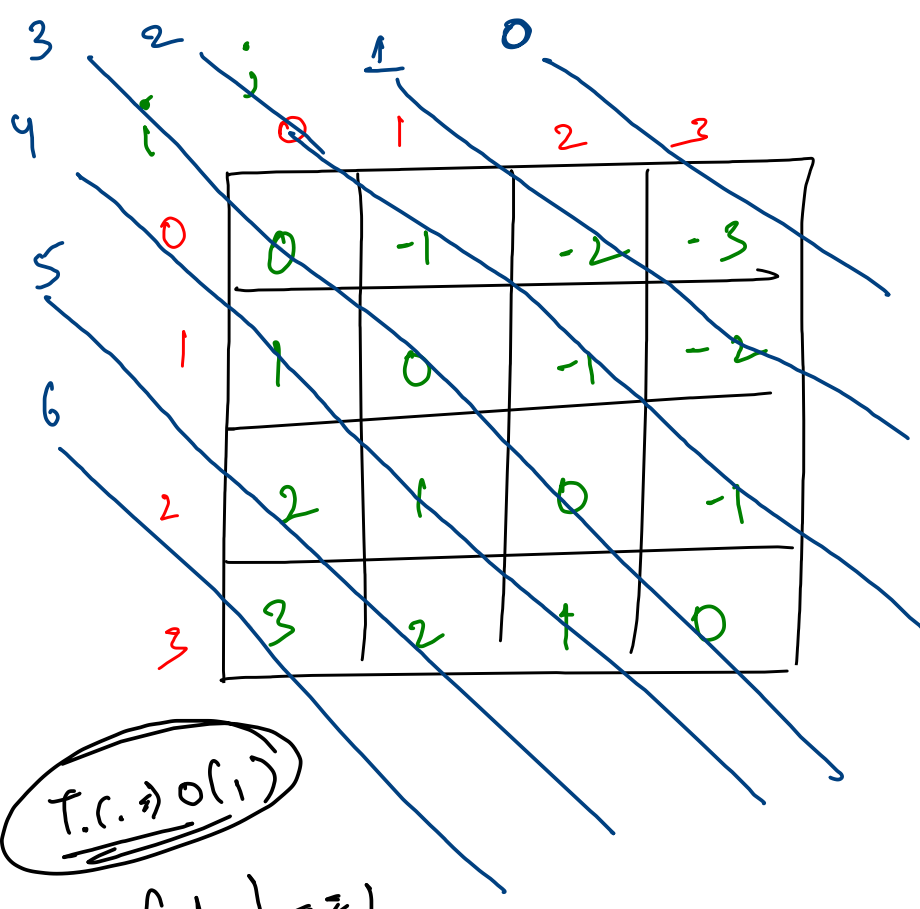


(2 (boolean)



false

Time => O(1)



Left Diag

$idx \Rightarrow i - j + n - 1$

$i - j$

$2n-1$

-3 -2 -1 0 1 2 3

0 1 2 3 4 5 6

ld = | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

T.C. $\Rightarrow O(1)$

ld[idx] == 1 → unsafe

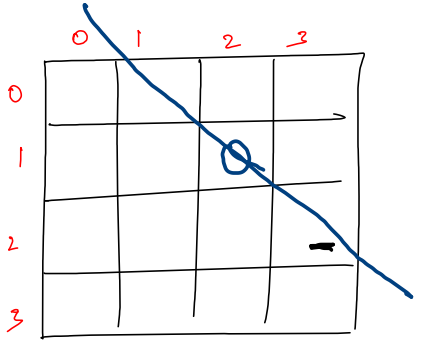
ld[idx] == 0 → safe

(i, j)

i = 1, j = 2

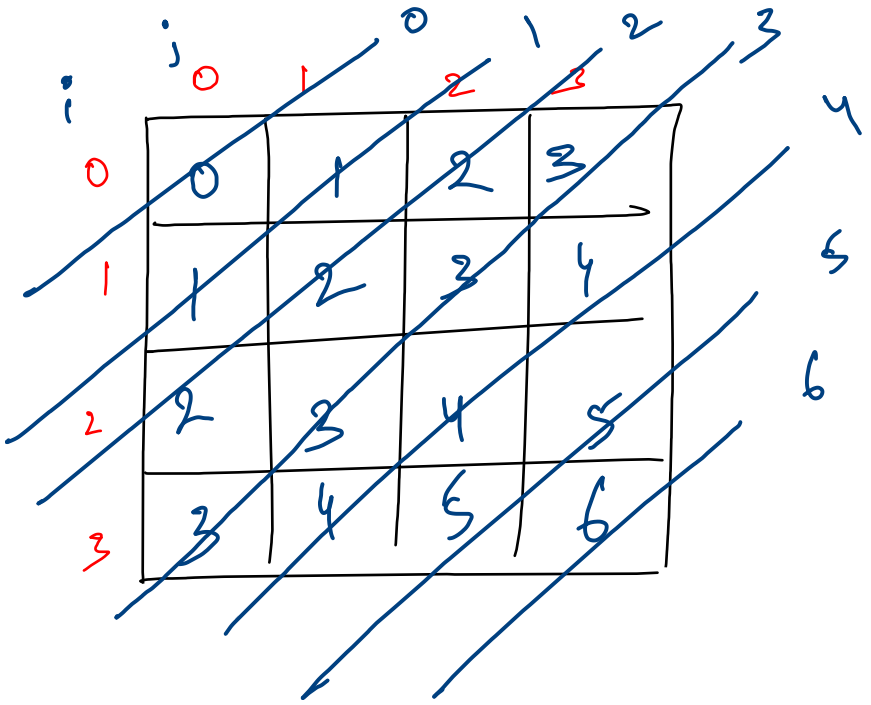
$idx \Rightarrow -1 + 4 - 1$

$\Rightarrow 2$



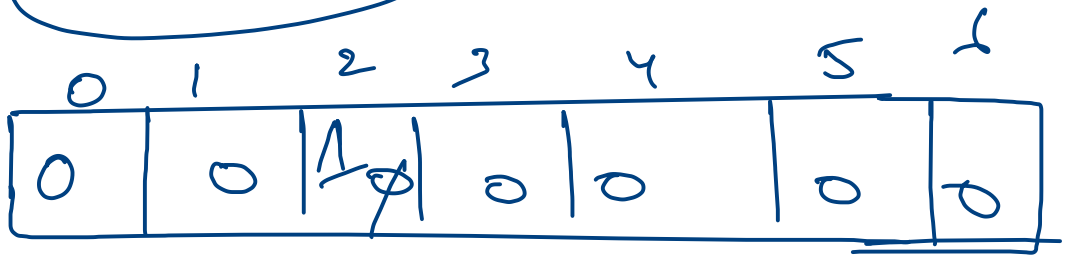
i = 2, j = 3

idx = 2



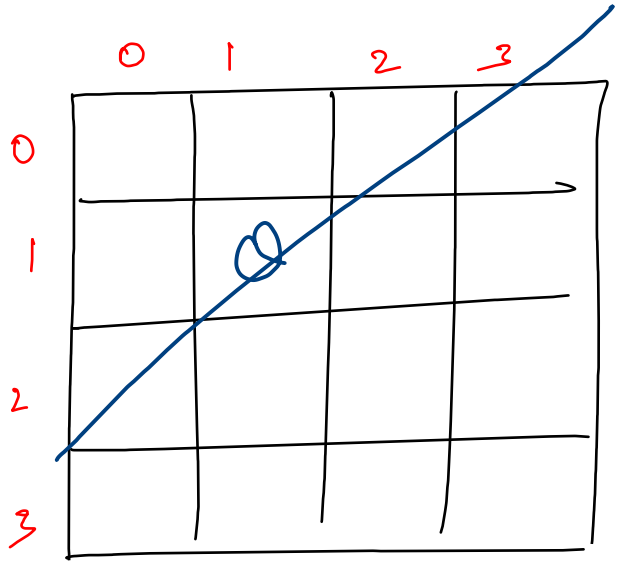
Right Diu

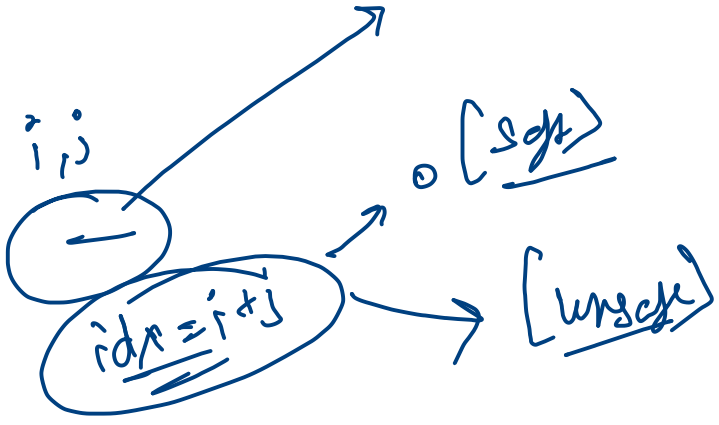
idx = i + j

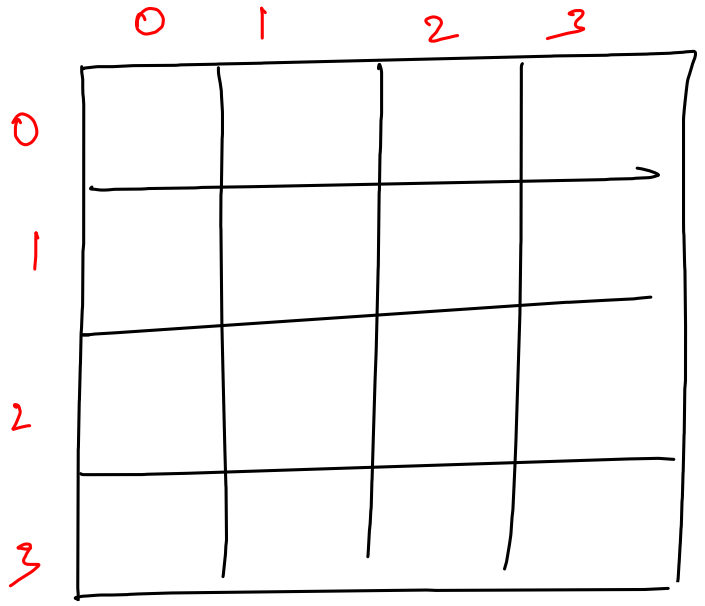


i = 1 , j = 1

i + j = 2







L1

n Queens

Space, Time <

B&B

Branch & Bound

Bit Manipulation

B1B

3 array

3 bitsets