Abbreviation 1 - Using bits:

p e p

p e p

0- 0 0 0 → p e p

1- 0 0 1 → p e 1

2- 0 1 0 → p 1 p

3- 0 1 1 → p 2

4- 1 0 0 → 1 e p

5- 1 0 1 → 1 e 1

6- 1 1 0 → 2 p

7- 1 1 1 → 3

~~pep~~

~~pe1~~

~~p1p~~

~~p2~~

~~1ep~~

~~1e1~~

~~2p~~

~~3~~

a b c

length = 3

a b c

String.length() - 1 → a b c

0 → 0 0 0

1 → 0 0 1 count = 1 → a b 1

2 → 0 1 0 count = 1 → a 1 c

3 → 0 1 1 count = 2 → a 2

4 → 1 0 0 count = 1 → 1 b c

5 → 1 0 1 count = 1 → 1 b 1

6 → 1 1 0 count = 2 → 2 c

7 → 1 1 1 count = 3 → 3

8

1<<3

8-1

1 1 1

count = 3

a b c d e

1 1 1 0 0

count = 3

if count != 0

char = append

a b c

cout=0 0 0 1 → a

cout=2 1 1 0 → 2c

1 0 1

a b c d e f

1 1 0 1 1 0 → 2c2f

cout=0 1 1 0 1 2

a b c d

1 1 1 1

cout=1 1 1 1 4

```java
StringBuilder res = new StringBuilder();
for(int i = 0; i < (1 << str.length()); i++) {
    int count = 0;
    for(int j = 0; j < str.length(); j++) {
        int k = str.length() - 1 - j; // bit in

        // check if kth bit is ON or OFF in 'i'
        int bm = 1 << k;
        char ch = str.charAt(j);
        if((i & bm) == 0) {
            // bit is OFF
            if(count == 0) {
                res.append(ch);
            } else {
                res.append(count);
                res.append(ch);
                count = 0; // reset count
            }
        } else {
            // bit is ON
            count++;
        }
    }
    if(count != 0) {
        res.append(count);
    }
    res.append("\n");
}
System.out.println(res);
```

No. of continuous 1's

length of string = 3

$2^3 - 1$

add 'enter' after an iteration

a b c = length of String = 3

0 → 0 0 0 → a b c \n

1 → 0 0 1 → a b 1 \n

2 → 0 1 0 → a 1 c \n

3 → 0 1 1 → a 2 \n

4 → 1 0 0 → 1 b c \n

5 → 1 0 1 → 1 b 1 \n

6 → 1 1 0 → 2 c \n

7 → 1 1 1 → 3 \n

$2^3 \equiv [1 << 3]$

calculate power of 2 in O(1) complexity.

j

0 → 3 − 0 → 3

1 → 3 − 1 → 2

2 → 3 − 2 → 1

3 → 3 − 3 → 0

Straight − j

3 time shift

0 0 0 0 1

0 0 0 1 0 → 2^1

0 0 1 0 0 → 2^2

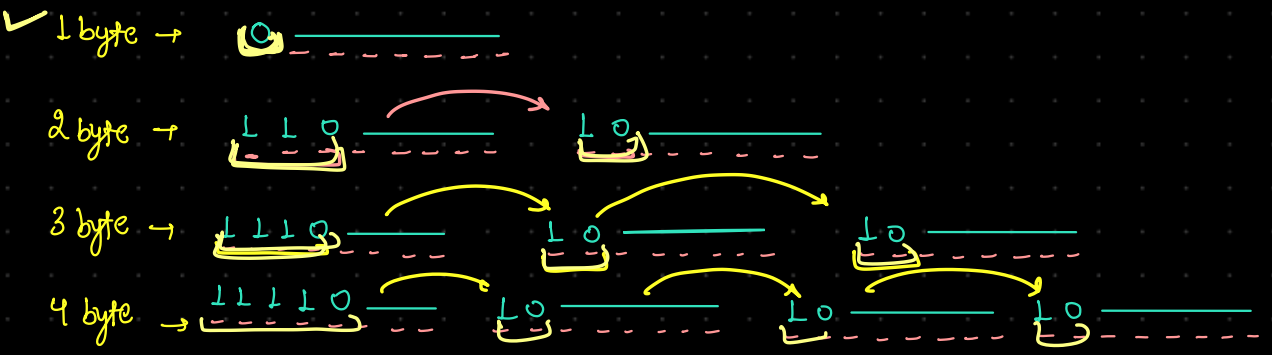
0 1 0 0 0 → 2^3

if (KC 14)

Utf-8 Encoding.

1 byte = 8 bits.

Utf-8 → 1 char = 1 to 4 byte



array → sequence of Utf

1 byte Number → 8 relevant bits

1 byte

1 1 0 | 1 0 1 1 → 2byte

1 0 | 0 1 0 0 1

1 1 1 0 | 0 1 0 → 3byte

1 0 | 0 1 0 1 1

1 0 | 1 0 0 1 0

0 | 0 1 0 1 1 0 1 → 1byte

1 1 1 1 0 | 0 0 1 → 4byte

1 0 | 0 1 1 0 1 0

1 0 | 1 1 0 1 1 0

1 0 | 1 0 1 1 0 1

num = 0b110 → 113

shifting

Array is following sequencing of Utf

0

110

1110

11110

10

1 1 0 | 0 1 1 1 0 → 2byte.

1 0 1 1 0 1 1 0

1 1 1 0 | 1 0 1 1 → 3byte.

1 0 1 0 1 1 1 0

0 1 0 1 0 1 0 → Return False.

1 0 1 1 1 0 1 1

Binary literals ⟶

Java 7

Feature → User can write number in binary form

Syntax ⟶

+ve Number:
- 0b binary
- OR
- 0B binary

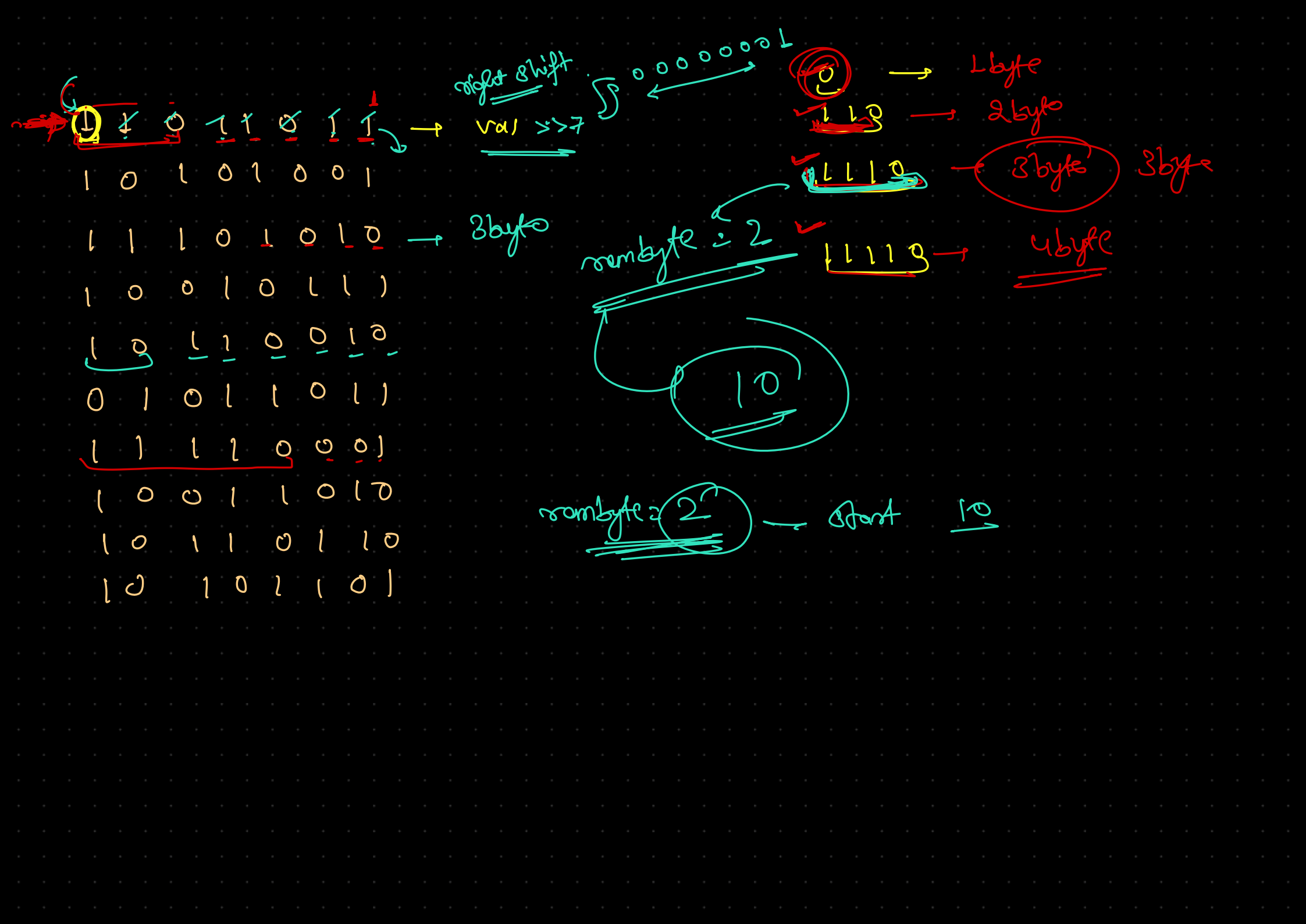
-ve Number:
- − 0b binary
- − 0B binary

Eg.

```
num = 0b101;
```

↳ binary form

num = 5

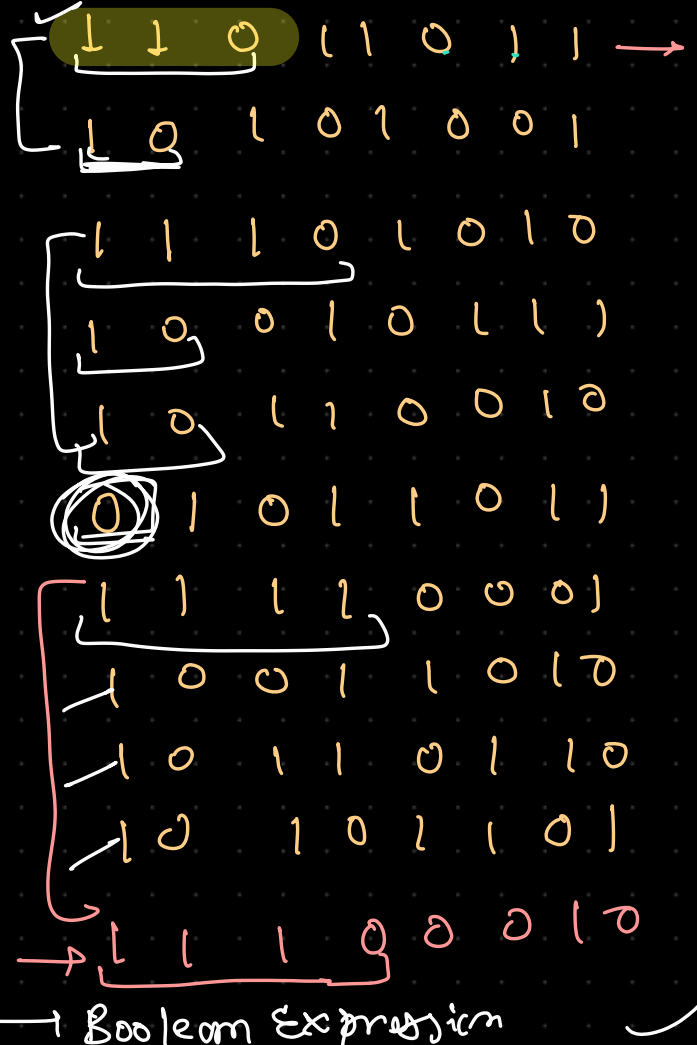

right shift
val >>7
1byte
2byte
3byte
3byte
4byte
numbyte : 2
10
numbyte : 2
start
10

```java
int remByte = 0;
for(int val : arr) {
    if(remByte == 0) {
        if((val >> 7) == 0b0) {
            // 1 byte character
            remByte = 0;
        } else if((val >> 5) == 0b110) {
            // 2 byte character
            remByte = 1;
        } else if((val >> 4) == 0b1110) {
            // 3 byte character
            remByte = 2;
        } else if((val >> 3) == 0b11110) {
            // 4 byte character
            remByte = 3;
        }
    } else {
        // check for remain byte i.e. it be
        if((val >> 6) != 0b10) {
            return false;
        }
        remByte--;
    }
}
return false;
```

return false; True.

remByte = 2

remByte == 0;



→ Boolean Expression

if(remByte == 0) return true

otherwise false

utf-8 →

0 → 1 byte

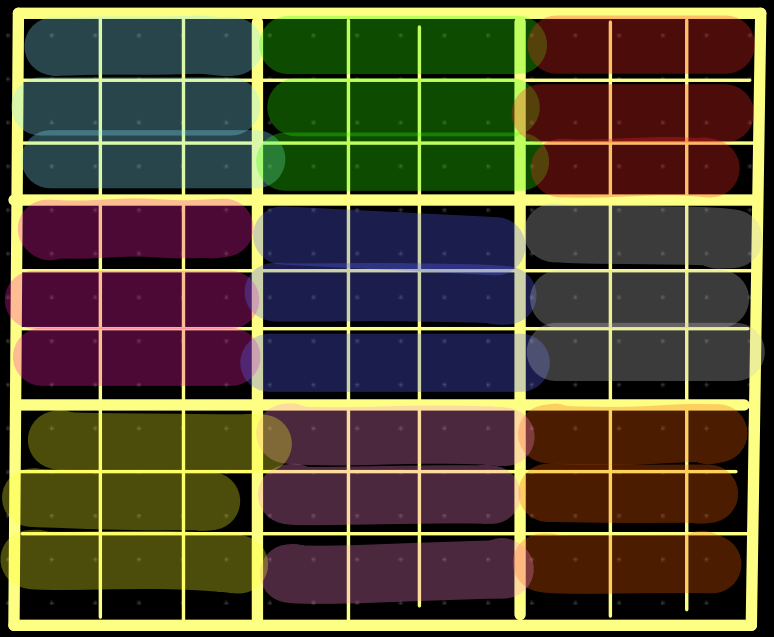
1 1 0 → 2 byte

1 1 1 0 → 3 byte

1 1 1 1 0 → 4 byte

Sudoku (Revise)

9x9 standard size



invalid:

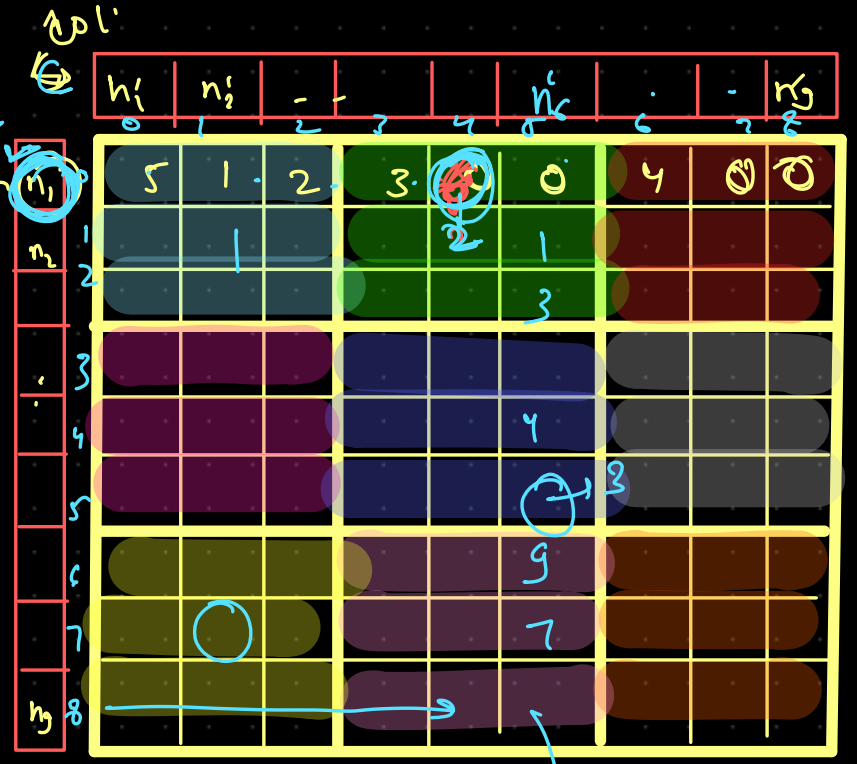
- avoid repitition of number in same Row - O(n) → O(1)
- avoid repitition of " " " col - O(n) → O(1)
- " " " " " sub matrix - O(n) → O(1)

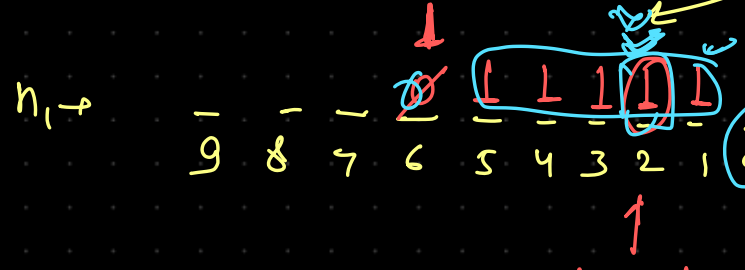
N-Queen → Branch and Bound

invalid

- avoid repitition of number in same Row - n → $O(n)$
- avoid repitition of " " " col → $O(n)$
- " " " " " " Sub matrix → $O(n)$ → $O(1)$

$O(1)$ $O(1)$





if 2nd bit is ON or OFF

validity optimised check