Bit manipulation

→ Binary representation of Negative No. is represented in 2's complement form

08 Range = $(-2^{n-1}$ to $2^{n-1}-1)$

1 = Negative
0 = Positive

09 formula to get 2's complement = $2^n - x$

eg $n=4$

10 range $(-2^3$ to $2^3-1)$

$(-8$ to $7) = 16$ Number.

eg $n=-3 = (1101)$

Storing a No.

↑ → 31st [sign bit]

11 Q Why 2's complement form?

① We have only one representation of zero

12 ② The arithmetic operation are easier to perform. Actually 2's complement form is derived from the idea of $0-x$.

01 ③ The leading bit is always 1.

02 → It is used to performe operation on binary operation.

→ Performance of the application will be improved.

03 → efficienty also will be implemented.

→ Bit manipulation operator are working with binary number direty.

04

convert to binary —

05 print (bin (18)) — 0b 10010 [bin(x)[2:]] ✓

print (bin (12)) — 0b 1100 bin(x).replace("0b", "")

06 Print (int ("0b 10010", 2)) — 18

print (int (" 0b1100", 2)) — 12

07

Bitwise operator.

→ It is a special operator that are existed in all the PL.

→ It is an efficient way to work with any application.

→ It is very fast when compared with other operator.

→ It requires linear time for execution (constant)

→ The result is returned in decimal format.

MARCH

02 '24

09th Week
062-304

SATURDAY

S	M	T	W	T	F	S
31					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

MAR 2024

binary to decimal, octal to decimal
↳ int (num, 2/8/16)
base जिसमे convert करेगा
↳ but input must bin str.

① bitwise AND — (&)
② bitwise OR — (|)
③ bitwise X-OR — (^) — if two bits are same = 0 else 1.
④ left shift — (<<) multiply by 2 [right में add zero]
⑤ right shift — (>>) divide by 2 — [left में add zero]
⑧ compliment — (~) -(n+1)

eg 11 n=5 $n<<1 = 1010 = (10)_{10}$
$n<<2 = 10100 = (20)_{10}$
101 12. $n<<3 = 101000 = (40)_{10}$

eg 01 n=5 $n>>1 = 010 = (2)_{10}$
$n>>2 = 001 = (1)_{10}$
101 02 $n>>3 = 000 = (0)_{10}$

eg 03 n=5 $(\sim n) = -6$ ✓

04 Advantage of bitwise operators

① speed ↑
05 ② space optimization
③ bit manipulation
06 ④ code simplification
⑤ readibility will be improved
07 ⑥ data encryption etc.

S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

APR 2024

4= 100 , odd → LSB = 1
5= 101 , even → LSB = 0

Power of 2 — to find power of 2 (or) not.

```
(a) def pow(n):
        if n==0:
            return False
        while n!=1:
            if n %2 !=0:
                return False
            n = n//2
        return True
```

```
(b) def pow(n):
        if n==0:
            return False
        return (n& (n-1) ==0)
```

One odd occuring

All number occurs an even numbers of times except One No. which occurs an odd No. of times.

```
(a) def fun(arr,n):
        for i in range(0,n):
            count=0
            for J in range(0, n):
                if arr[i]==arr[J]:
                    count +=1
            if (count %2 !=0):
                return arr[i]
        return -1
```

$TC=O(n^2)$

```
(b) def fun(arr):
        res=0
        for i in arr:
            res = res ^ i
        return res.
```

eg arr=[4,3,4,4,4,5,5,3,3]

3 ✓ left out

⇒ n ^ n = 0

Two odd occuring

```
(a) def fun(arr):
        for i in arr:
            count=0
            for J in arr:
                if i==J:
                    count +=1
            if (count%2)!=0:
                print(i, end="")
```

$O(n^2)$

(%2 !=0)

```
(b) def fun(arr):
        xorr, res1, res2 = 0
        for i in arr:
            xorr = xorr ^ i
        sn = xorr & ~(xorr-1)
        for i in arr:
            if i&sn !=0 : res1=res1^i
            else : res2=res2^i
        Print(res1,res2)
```

$2^n \rightarrow (1 << n)$

→ '1' [that bit is 1]
Search from right side

Check the kth bit is set or not.

08 i/p — n = 5, k = 1   101 ✓   i/p — n = 8, k = 2   1000 ✗
o/p — yes   o/p — No

09

m1
```
def fun(n,k)
    if n & (1 << (k-1)):
        Print ('Set')
    else:
        Print ('Not set')
```
→ Leftshift
→ K aswa हो जाएगा।

n = 5 — 0 0   0101
1<<(k-1) — 0 0   0100
4 &
4   0 0   100

m2
```
def fun (n,k):
    if ((n >> (k-1)) & 1):
        Print ('Set')
    else:
        Print ('Not Set')
```
→ right shift.
→ K को लिख सकते हैं

n = 5 — 0 0   101
n>>(k-1) — 0 0   001
1 — 0 0 0   0 01

03

Count set bits

```
def count(n):
    res = 0
    while n:
        if (n%2 == 1):
            res = res + 1
        n = n//2
    return res
```
(n&1)
θ(n)

Brian Kernigan's Algorithms

```
def count (n):
    res = 0
    while n:
        n = n & (n-1)
        res = res + 1
    return res
```
θ(set bits)

n = 40 : 101000
(n-1) = 39 : 100111
n &(n-1) : 32 = 100000

n=32 : 100000
n-1 = 31 : 011111
n&(n-1) = 0 : 000000

→ { x ^ 0 = x
x ^ x = 0 }

Find the only odd. → Other No. must appear even, No. of times except one odd

i/p – l=[10, 30, 30, 10, 30, 30, 20]
o/p – 20

i/p – [10, 10, 20, 30, 30, 20, 40]
o/p – 40

i/p → l= [10, 10, 10, 10, 10, 20, 20]
o/p – 10

M-1

```
def fun(l):
    res = NoNe
    for x in l:
        count = l.count(x)
        if count %2 != 0:
            res = x
            break
    return res
```

M-2

```
def fun(l):
    res = 0
    for x in l
        res = res ^x
    return res
```

Powers Set using Bitwise

$\overset{0\ 1\ 2}{[1\ 2\ 3]}$

N = 1, 2, N = 3→8
N = 2, 4, N = 4, 16

M-1

```
def fun(s):
    n= len(s)
    Psize = (1 <<n)
    for i in range(Psize):
        for J in range(n):
            if ((i & (1<<J)) != 0):
                print(S[J], end = '')
        print()
```

o/p— ""
"a"
"b"
"ab".

n=3

0	1	2	
2	1	0	
0	0	0	→ []
0	0	1	→ [1]
0	1	0	→ [2]
0	1	1	→ [1,2]
1	0	0	→ [3]
1	0	1	→ [1,3]
1	1	0	→ [2,3]
1	1	1	→ [1,2,3]

M-2 → यदि n= 3 होगा तो 0 में take 4, → 0
don't take 4 → 1.
1 में take 2, don't take → 2
2 " take 1, don't take → 1

यदि n=4 होगा तो 0 में take 8, don't take 8}

{0 → don't take
1 → take it}

MARCH
06 '24
10th Week
066-300
WEDNESDAY

S	M	T	W	T	F	S
31					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

MAR 2024

~~M-2 for power set~~

~~l=[]~~

~~for i in range(len(1<<n)-1):~~

~~for J in~~

~~if (num & (1<<i))~~

~~l.append(nums[i])~~

Right most different bit

```
C=0
while m!=0 or n!=0:
    C +=1
    if m%2 != n%2:
        return C
    m//=2
    n//2 = 2
return -1
```

✓

Method② for finding power set

```
def power(s):
    n=len(s)
    power_set=[]
    for i in range (1<<n)
        Subset = [S[J] for J in range(n) if (i& (1<<J))]
        power-set.append(Subset)
    return power-set
```

✓

S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

Set the ith bit to 1 → N|(1 << bit position)

Set the ith bit to 0 → N & ~(1 << Bit position)

Set the ith bit.

जो भी No. दिया है इसको ith position से left shift करेंगे the OR Operation
08 perform करेंगे।

eg n = 9 , i=2

```
   2/10
 - 10 01
   01 00   left shift by 2
   -----
   1 1 0 1
```

09

→ N or (1 << i)

10 N | (1 << i)

If it is set then you will get same No. if Not then do it.

11

```
→ def (n, i):                      def (n,i):
12     newnum = n | (1 << ith          newnum = n & (~1 << i)
       return newnum                   return newnum.
```

01 ✓

Clear / toggle the ith bit.

turn the bits to 0,
02 if it is 0 → Keep it 0.

```
def (n,i):
03   newnum= n & (~(1 << i))
```

04 Toggle — n ^ (1 << i)

05

06

07

Min bit flip to convert number.

Start = 10 , goal = 7
↓ 1010 ↓ 0111

$$= \begin{array}{r} 1010 \\ \wedge\ 0111 \\ \hline 1101 \end{array} \rightarrow \boxed{3}$$

~~Count~~ = 0

```
~~for~~ def flip(n1,n2):
        diff = n1 ^ n2
        count = 0
        while diff:
            count = count + diff & 1
            diff >>~~=~~ 1
        return count
```

XOR of Nos in Given range.

N%4 == 0	N
N%4 == 1	1
N%4 == 2	N+1
N%4 == 3	0

```
def f1(n):
    if (N%4 == 1): return 1
    elif (N%4 == 0): return N
    elif (N%4 == 2): return N+1
    else (N%4 == 3): return 0.
```

TC— O(1)
SC— O(1)

```
def f2(L,R):
{ return f1(L-1) ^ f1(R)
}
```