Bit Manipulation

Sunday, 12 November 2023 2:27 AM

① Binary Representation of Negative Numbers
- Negative numbers are represented in 2's complement form.
- Range of numbers: $[-2^{n-1}$ to $2^{n-1}-1]$ (n is number of bits)
- Steps to get 2's complement: (a) Invert all bits (b) Add 1

Direct formula = $2^n - x$

Why 2's complement form? → other options → Signed bit, → 1's complement
1) We have only one representation of zero.
2) The arithmetic operations are easier to perform. Actually 2's complement form is derived from the idea of 0−x.
3) The leading bit is always 1. ($2^{n-1}-1$ → leading bit always 0 for +ve numbers)

Note: 1) The first bit in a signed representation is the sign of number, 0 for non-negative numbers and 1 for negative numbers, and the remaining (n−1) bits contain the magnitude of the number.
2) Two's complement is used for negative numbers.
3) In an unsigned representation, only non-negative numbers can be used, but the upper bound for values is higher.
— contains integers between 0 to 2^n-1.
Connection between two representation: A signed −x equals to an unsigned 2^n-x.
4) If a number is larger than the upper bound of bit representation, the number will overflow.
— In a signed representation, next no. after $2^{n-1}-1$ is -2^{n-1}.
— In an unsigned representation, next no. after 2^n-1 is 0.

② Bitwise Operators in Python

	o/p
print (bin(18))	0b10010
print (bin(12))	0b1100
print (int("0b10010",2))	18

Special prefixes for all bases:
→ for integer/decimal representation → no prefix
→ for binary representation → 0b
→ for octal → 0O
→ for hexadecimal → 0X

Bitwise AND (&) — Leftshift operator (<<)
Bitwise OR (|) — Rightshift operator (>>)
Bitwise XOR (^) — Bitwise Not (~)

Operator	Operation	Result
XOR	X ^ 0s	X
XOR	X ^ 1s	~X
XOR	X ^ X	0

Operator	Operation	Result
AND	X & 0s	0
AND	X & 1s	X
AND	X & X	X

Operator	Operation	Result
OR	X | 0s	X
OR	X | 1s	1s
OR	X | X	X

Here, 0s or 1s mean a sequence of 0 or 1.

eg: X=6 ⇒ 0110 — 0110 & 0000 = 0000 → 0; 0110 & 1111 = 1001 → ~X; 0110 & 0110 = 0110 → X

eg: X=7 ⇒ 0111 — 0111 & 0000 = 0000 → 0; 0111 & 1111 = 0111 → X; 0111 & 0111 = 0111 → X

eg: X=5 ⇒ 0101 — 0101 & 0000 = 0000 → 0; 0101 & 1111 = 0101 → X; 0101 & 0101 = 0101 → X

③ Get Bit
— find the bit at a particular position of given number N.
— find Bitwise AND of given number and 2^{K-1} (1<<(K-1))
— if value is 1 then bit is set else unset.

(num & (1<<K-1)) != 0 or ((num>>(K-1)) & 1) != 0
↳ at K^{th} position ↳ at K^{th} position

Set Bit
— method is used to set the bit at a particular position (say K).
— idea is update the value of given number N to the Bitwise OR of N and 2^{K-1} (1<<(K-1)).
— if value is 1 then bit is set.

(num | (1<<K-1))

Clear Bit
— method used to clear bit at K^{th} position of a number N.
— update the given number N to Bitwise AND of N and complement of 2^{K-1}, ie ~(1<<(K-1)).
— mask = ~(1<<(K-1))
return num & mask

Toggling a bit at K^{th} position
— use XOR because 0^0=1^1=0 and 0^1=1^0=1
— if two bits are different then XOR operator returns a set bit (1) else it returns an unset bit (0).

num = num ^ (1<<(K-1))

Inverting every bit of a number / 1's complement
— using NOT operator ⇒ return (~num)

2's complement of a number
— 1's complement + 1 or using '-' operator
(~num+1) or (-num)

Stripping of lowest set bit (Brian Kernighan)
X = X & (X-1)
— (X-1) inverts all bits till it encounters the lowest set '1' and also inverts the lowest set '1'
— After adding X and (X-1), we get lowest set bit stripped.

Getting the lowest set bit
num = num & (-num)
— If we AND original number X with 2's complement which is −X, we get lowest set bit.
eg: X = 12 ⇒ 1100 — 1's comp → 0011 ⇒ 2's comp → 0100 — 1100 & 0100 = 0100

Divide by 2 using right shift operator
num >> 1

Multiply by 2 using left shift
num << 1

Bit Tricks

① Clearing all bits from LSB to i^{th} position
mask = ~((1<<i+1)-1)
X & = mask

eg: N=29 (00011101) and clear LSB to 2nd bit (4 bits)
→ mask = 1<<4 ⇒ 16 (00010000)
→ mask = 16-1 ⇒ 15 (00001111)
→ mask = ~mask ⇒ 11110000
N & mask ⇒ 00010000

Logic: to clear all bits from LSB to i^{th} bit, we have to AND n with mask having LSB to i-th bit 0.
— to obtain such mask, first left shift 1 i times.
— Now, if we minus 1, all bits from 0 to i-1 becomes 1 and remaining bits become 0.
— Now, simply taking complement of mask to get all first i bits to 0 and remaining to 1.

② Clearing all bits from MSB to i-th bit
mask = (1<<i)-1
X & = mask

③ Multiply/Divide by 2 ⇒ left shift/right shift by 1

④ Uppercase to lowercase alphabet
— Note: if we set 5^{th} bit of uppercase character, then it will be converted to lowercase character.

A → 01000001 — a → 01100001
Z → 01011010 — z → 01111010

— mask = 00100000 represent space (' ')
Character ored with mask will convert to lowercase
Ch = ch | ' ';

⑤ Lowercase to uppercase alphabet
— mask = 11011111 represents underscore ('_')
Ch = ch & '_';

⑥ Count set bits

Brian Kernighan's Algorithm
→ Subtracting 1 from decimal number flips all the bits after the rightmost set bit (which is 1) including rightmost set bit.
— So if we subtract 1, and do it bitwise and with itself (n&(n-1)), we unset rightmost bit.
— If we do n&(n-1) in a loop and count the number of times the loop executes, we get set bit count.

① Compute XOR from 1 to n
if n%4 is 0 : return n
if n%4 is 1 : return 1
if n%4 is 2 : return n+1
if n%4 is 3 : return 0

② Count of numbers (x) smaller than equal to n such that n+x = n^x
mathematical trick: count = pow(2, count of zero bits)
we know → n+i = (n^i) + (n&i)
— first values of i such that n&i=0.
— we can count unset bits in the binary representation of n.
— i must unset all set-bits of n.
— If k^{th} bit is set in n, k^{th} bit in i must be 0, else k^{th} bit can be 0 or 1.
— Hence total combinations are $2^{(count\ of\ unset\ bits\ in\ n)}$

③ How to check if a number is power of 2?
If a no. is power of 2, then bitwise AND of N and N-1 will be 0.
so, just check these two conditions, ① if N is not 0.
② N&(N-1) == 0.
return n and (not (n&(n-1)))

④ Find XOR of all subsets of a set
— ans is 0 if given set has more than one element.
— for set with single element, answer is value of single element.

⑤ Quickest way to swap two numbers
a ^= b
b ^= a
a ^= b
a = a^b
b = b^(a^b) = a
a = (a^b)^(a) = b

⑥ Find most significant set bit
K = int(math.log(n,2))
return 1<<k

⑦ Check if a number has bits in an alternate pattern
— compute bitwise XOR of n and n>>1
— If n has alternate pattern, then n^(n>>1) operation will produce a number having all bits set.