Bitwise Operations Tips and Examples

May 19, 2020

Contents

1 Intro

My attempt to understand and exemplify uses of bitwise operations with realistic examples.

Not all languages support bitwise operations on non-integer numbers. In fact, looks like the majority don't, and just a few do.

I'm mostly concerned with these languages (alphabetical order):

- C
- Haskell
- JavaScript
- Python
  - Operands must be integers:
    - `https://docs.python.org/3/reference/expressions.html#binary-bitwise-operations`

- Ruby
- Scheme
- TypeScript

Languages supporting bitwise on decimals and integers:

- JavaScript

Languages supporting decimals only on integers:

- Python
- Ruby

2 parseInt

When a language supports bitwise operations, we can parse an int out of a decimal using the XOR.

JavaScript:

```
parseInt(3.14, 10);
// 3

0 ^ 3.14
// 3

(0 ^ 3.14) === (parseInt(3.14, 10))
// true
```

3 find int odd times

- https://www.codewars.com/kata/54da5a58ea159efa38000836/train/javascript

```
const l = console.log.bind(console);

/**
 * Find number that appear an odd number of times.
 *
```

```
 * ASSUME: There will always be only one integer that appears
 * an odd number of times.
 *
 * @param {array<number>} arr
 * @return {number}
 */
const findOdd = arr => arr.reduce((acc, n) => acc ^ n);

l(findOdd([20, 1, -1, 2, -2, 3, 3, 5, 5, 1, 2, 4, 20, 4, -1, -2, 5]));
// → 5;

l(findOdd([1, 1, 2, -2, 5, 2, 4, 4, -1, -2, 5]));
// → -1

l(findOdd([20, 1, 1, 2, 2, 3, 3, 5, 5, 4, 20, 4, 5]));
// → 5

l(findOdd([10]));
// → 10

l(findOdd([1, 1, 1, 1, 1, 1, 10, 1, 1, 1, 1]));
// → 10

l(findOdd([5, 4, 3, 2, 1, 5, 4, 3, 2, 10, 10]));
// → 1
```

NOTE: This is my own explanation.
XORing x to itself always yields 0.

```
1 ^ 1 = 0
1 ^ 1 ^ 1 ^ 1 = 0
```

XORing x to 0 yields x.

```
1 ^ 0 = 1
5 ^ 0 = 5
```

If we XOR x an even number of times, the result is always 0. It means an even number of x's cancel themselves.

For this challenge, since it is ASSUMED that there will be only one number that appears an even number of times, we can use this bitwise XOR technique to obtain the answer.

```
1 ^ 1 ^ 3 = 3
1 ^ 2 ^ 1 ^ 2 ^ 3 = 3
```

The XOR operator has the associative and commutative properties. Thus, the order of the numbers in the array doesn't matter for this case.

- `https://en.wikipedia.org/wiki/Exclusive_or#Properties`

Of course, the reducer's accumulator is cleverly used here too!

3.1 Additional Explanations

These explanations are from the comments found on the challenge page. I DID NOT write them. Just reworded some stuff or added extra details for my own understanding.

^ means XOR operation.

Capital letters as A or B or X are the result of a xor operation.

```
## 1. Truth table
-----------------

0 ^ 0 = 0
0 ^ 1 = 1
1 ^ 0 = 1
1 ^ 1 = 0
```

This is usefull to undertand second thing below.

```
## 2. Properties
-----------------

A ^ A = 0 ( ex: 10 ^ 10 = 0 )
A ^ 0 = A
```

(ps: we don't need other properties to undertand)

```
## 3. Associativity
--------------------

a ^ b ^ c = a ^ c ^ b
or even
a ^ b ^ c ^ d = a ^ c ^ d ^ b
```

So this means that the priority order of operations can be changed, this is not mandatory to start by doing a ˆ b operation.

```
## REAL EXAMPLE 1
-----------------

[10, 3, 20, 10, 3, 20, 10]
here 10 is the number that is repeated odd times

This solution will iterate like this.
- 10 ^ 3 = A (it's 9 but we dont need to know real results)
- A ^ 20 = B it's the same as 10 ^ 3 ^ 20 so B = 10 ^ 3 ^ 20 ..and so on
- 10 ^ 3 ^ 20 ^ 10. At this moment we can use associativity, change
the order or prio operations
so we can write 10 ^ 10 ^ 3 ^ 20, now use the properties (A ^ A = 0)
so 10 ^ 10 = 0 ... then 0 ^ 3 ^ 20. Again use the property (A ^ 0 = A)..
so 0 ^ 3 ^ 20 = 3 ^ 20. we continu iteration ...

- 3 ^ 20 ^ 3 .. Again use associativity and properties, the result here is 20
- 20 ^ 20 = 0, then last iteration
- 0 ^ 10 = 10 !

As you see the behaviour is that: if at a time we meet/encounter a number
that's already IN previous  XOR operations .. like:
[a] ^ b ^ c ^ [a] the reapeated number is somehow canceled or removed.
```

Thats why XOR operation can resolve this kind of problem. But only with this particular set prerequisites.

Even numbers will eventually be offset, leaving only an odd number int.

4 Links and Resources

- Interesting use cases for JavaScript bitwise operators (Logrocket)
- Bitwise operators — Facts and Hacks (Medium)

4.1 Ryonagana Archdark

Links shared by Archdark:

- `https://www.gamedev.net/reference/articles/article1563.asp`

- http://graphics.stanford.edu/~seander/bithacks.html
- https://www.cprogramming.com/tutorial/bitwise_operators.html
- https://catonmat.net/low-level-bit-hacks
- http://www.codeproject.com/KB/cpp/bitbashing.aspx
- http://www.eskimo.com/~scs/cclass/int/sx4ab.html
- http://www.cs.utk.edu/~vose/c-stuff/bithacks.html
- http://www.somacon.com/p125.php
- http://www.fredosaurus.com/notes-cpp/expressions/bitops.html
- http://goanna.cs.rmit.edu.au/~stbird/Tutorials/BitwiseOps.html
- http://www.custard.org/~andrew/programming/bitwise/
- http://www.codeproject.com/KB/cpp/Bitwise_Operation.aspx
- http://bits.stephan-brumme.com/
- http://en.wikipedia.org/wiki/Bit_manipulation
- http://www.hackersdelight.org/HDcode.htm
- http://www.arduino.cc/playground/Code/BitMath
- http://www.fredosaurus.com/notes-cpp/expressions/bitops.html