Bitmasks & iterative complete search

Agenda

1. Bitwise operations
    - AND
    - OR
    - XOR
    - NOT
    - Left shift
    - Right shift
2. Bit Masking
3. Iterative complete search
4. Practice problems

1. Bitwise operations

Bitwise operations are used to manipulate individual bits of binary numbers. There are six bitwise operators in C++:

- Bitwise AND (&)
- Bitwise OR (|)
- Bitwise XOR (^)
- Bitwise NOT (~)
- Left shift (<<)
- Right shift (>>)

Notes:

1. All the bitwise operations are commutative, meaning that:
   - (A operation B) equals (B operation A), and
   - ((A operation B) operation C) equals ((C operation B) operation A) etc..
2. Bitwise operations have the lowest order in all available operations in C++ and all of them have the same priority.

- **Bitwise AND (&)**

  The bitwise AND operator returns a 1 in each bit position where both operands have a 1, otherwise, it returns 0.

  Example:

  a = 60 // 0011 1100
  b = 13 // 0000 1101
  c = a&b // 0000 1100

  Truth table:

A	B	A & B
0	0	0
0	1	0
1	0	0
1	1	1

- **Bitwise OR (|)**

  The bitwise OR operator returns a value where each bit position is set to either a one or zero depending on whether one or both operands have a one in that position.

  Example:

  a = 60 // 0011 1100
  b = 13 // 0000 1101
  c = a | b // 0011 1101

  Truth table:

A	B	A | B
0	0	0
0	1	1
1	0	1
1	1	1

- **Bitwise XOR (^)**

  The bitwise XOR operator returns the value where each bit position is set to one if only one of the two corresponding bits is one (It's like asking are the bits different?).

  Example:

  a = 60 // 0011 1100
  b = 13 // 0000 1101
  c = a^b // 0011 0001

  Truth table:

A	B	A ^ B
0	0	0
0	1	1
1	0	1
1	1	0

  Notes:

  A ^ A = 0
  A ^ B ^ A = B

  Problem:

  Given N numbers as input where every number will occur an even number of times except for 1 number, that 1 number will appear an odd number of times. Find that number.

- **Bitwise NOT (~)**

  The bitwise NOT operator inverts all the bits of an operand.

  Remember that we are dealing with SIGNED numbers that means that the most significant bit (the last bit on the left) determines the sign so, when using the NOT operation, positive numbers become negative and vice versa.

  Example:

  a = 60 // 0011 1100

  a = ~a (-61) // 1100 0011

  Truth table:

A	~A
1	0
0	1

- **Bitwise left shift (<<)**

  The left shift operator shifts all the bits of an operand to the left by the specified number of positions and fills the empty spaces with zeros.

  Left shifting a number by n bits is equivalent to multiplying it by 2^n. This is because when we shift a binary number to the left, we are essentially moving each digit to the left and adding a zero at the rightmost position. For example, if we have the binary number 101 (which represents the decimal number 5), and we shift it to the left by 2 bits, we get 10100 (which represents the decimal number 20). In this case, we have effectively multiplied the original number by 2^2 (which is equal to 4). This is because each digit in a binary number represents a power of two, with the rightmost digit representing 2^0, the next digit representing 2^1, and so on.

  Therefore, when we add a zero at the rightmost position during a left shift, we are effectively multiplying the original number by an additional power of two.

  Example:

```
a = 60        // 0011 1100
c = a << 2    // 1111 0000
```

- **Bitwise right shift (>>)**

  The right shift operator shifts all the bits of an operand to the right by the specified number of positions and fills the empty spaces with zeros.

  Right shifting a number by n bits is equivalent to dividing it by 2^n. This is because when we shift a binary number to the right, we are essentially dividing it by 2. For example, let's consider the binary number 110101. If we shift it one bit to the right, we get 011010. This is equivalent to dividing the original number (53 in decimal) by 2. Similarly, if we shift the binary number two bits to the right, we get 001101, which is equivalent to dividing the original number by 2^2 or 4.

  In general, when we right shift a binary number n bits, we are effectively dividing it by 2^n. This is because each bit that is shifted represents a power of two (starting from 2^0 for the least significant bit), and shifting it to the right reduces its value by half.

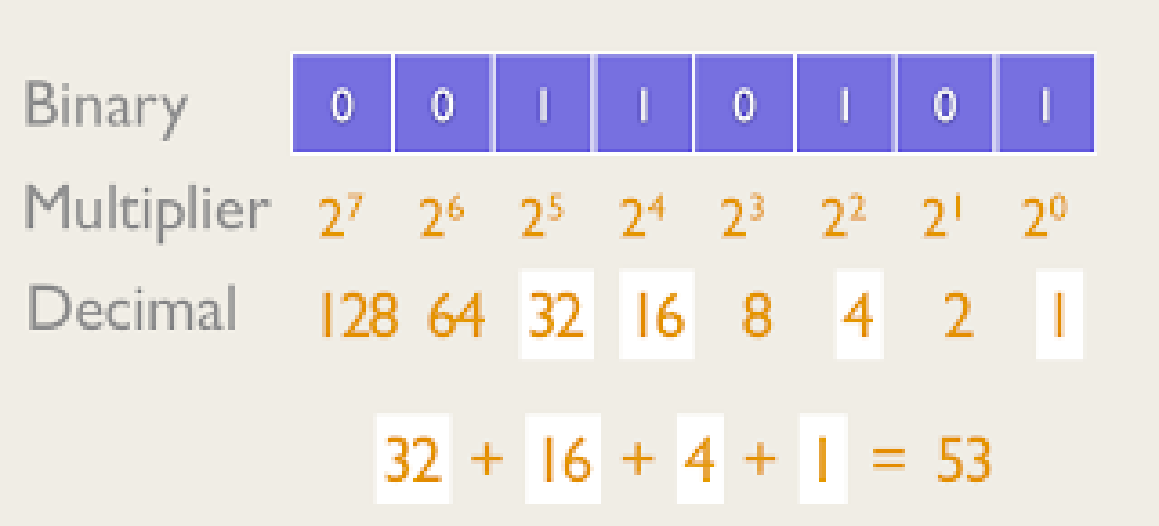
  Example:

  a = 60 // 0011 1100

  c = a >> 2 // 0000 1111

2. Bit Masking

- Bit masking is a technique used in computer programming to manipulate individual bits or groups of bits.
- Bits are numbered from RIGHT to LEFT starting from index 0 (2^0).
- C++ integers can hold 32 bits (0 to 31), long long can hold 64 bits (0 to 63).



a. Check if a bit is set (equal 1)

```
bool isSet(int mask, int bit) {
      return mask >> bit & 1;
}
```

or:

```
bool isSet(int mask, int bit) {
      return mask & (1 << bit);
}
```

b. Turn on a bit (set it = 1)

```
int turnOn(int mask, int bit) {
      return mask | (1 << bit);
}
```

c. Turn off a bit (set it = 0)

```
int turnOff(int mask, int bit) {
      return mask & ~(1 << bit);
}
```

d. Toggle a bit

```
int toggle(int mask, int bit) {
      return mask ^ (1 << bit);
}
```

e. Count the number of ones

```
int countOnes(int mask) {
      return __builtin_popcount(mask);
}
```

3. Iterative complete search

Complete search (aka brute force) is a method for solving a problem by traversing the entire search space in search of a solution.

A simple example of that is printing all numbers between 1 and 100 which are divisible by 5. The brute force solution is to try each number between 1 and 100 and check if it's divisible by 5 or not. Sometimes, the search space is not that easy to be implemented in one single for loop.

For example:

Let's say you have some unique numbers and a target sum, and you want to get the number of subsets that sum up to the target.

```
Numbers: 1, 5, 2, 7, 3, 9
Target: 6
Number of valid subsets: 2
{1,5}, {1,2,3}
```

But how to generate all the different subsets?

To solve this, we will represent each subset from that array as a binary number (mask) where each bit will represent an index, if the bit is lit (1) that means that the index is included in the current subset, and if the bit is not lit (0) that means that the index is NOT included in the subset.

Example:

All the subsets of the array {1,2,3} are:

{}	000
{3}	001
{2}	010
{2,3}	011
{1}	100
{1,3}	101
{1,2}	110
{1,2,3}	111

Our array has only 3 indices so we only need 3 bits to represent any possible subset. From that, we can deduce that the number of subsets in any array equals 2^N where N is the size of the array.

```
for (int mask = 0; mask < (1 << n); mask++) {
   int sum = 0;
   for (int i = 0; i < n; i++) {
       if((mask >> i) & 1)
           sum += nums[i];
   }
   if (sum == target)
       counter++;
}
```

4. Practice problems

- Preparing Olympiad
- Raising Bacteria